Clube Delphi + PHP

DevMedia GROUP

Ano 7 · Edição 94 · R$9,90

Coluna: Ask the Expert

Expert
Implemente compactação, download em múltiplos pacotes e resources em suas aplicações com técnicas avançadas de Streams – Parte 2

Expert
Saiba mensurar o tamanho de seu software e estimar o tempo que levará para ficar pronto

Mão na Massa
ClientDataSet: Veja como tratar erros automaticamente gerados pelo servidor Firebird

DataSetProvider

Saiba tudo sobre um dos principais componentes de acesso a dados no Delphi

Mini-Curso
Veja como criar um sistema on-line de controle para uma vídeo-locadora, com ASP.NET – Parte 2

Desenvolvendo uma Aplicação Completa com PocketStudio
Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 3

Coluna: QuickUpdate

ISSN 1517990-7

NA WEB >> Confira no portal ClubeDelphi PLUS um mini-curso sobre criação de um site com ASP.NET e SQL Server 2005 Express

- **Easy Delphi**
  Veja como criar um Editor de Textos e aprenda a utilizar na prática os principais controles da VCL
- **Delphi for PHP**
  Como aplicar conceitos de POO em aplicações PHP? - Parte 2
- **Easy Delphi**
  Envie e-mails automaticamente com o Delphi

Seu potencial. Nossa inspiração.
Microsoft

2008 Microsoft Corporation. Todos os direitos reservados. Microsoft, Visual Studio, o logo do Visual Studio e "Seu potencial. Nossa

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Expert

08 – Streams
Implemente compactação, download em múltiplos pacotes e resources em suas aplicações com técnicas avançadas de Streams – Parte 2
[*Gustavo Chaurais*]

Expert
Engenharia de Software

16 – Análise de Pontos de Função
Saiba mensurar o tamanho de seu software e estimar o tempo que levará para ficar pronto
[*Carmo Crêdiney de Melo e Marco Antônio Pereira Araújo*]

Mão na Massa

24 – Usando todo o poder do TDataSetProvider
Usufrua de todos os recursos do DataSetProvider em suas aplicações
[*Adriano Santos*]

Mão na Massa

32 – ClientDataSet
Automatizando o tratamento de Erros
[*Rodrigo Lazoti*]

Web
Mini-Curso
Mão na Massa

36 – Controle on-line de vídeo-locadora - Parte 2
Veja como criar um sistema on-line de controle para uma vídeo-locadora
[*Maikel Marcelo Scheid*]

Tutorial
Mini-Curso

42 – Desenvolvendo uma Aplicação Completa com PocketStudio
Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 3
[*Ricardo C. Boaro*]

Easy

50 – Envio de E-mails com componentes da paleta Indy
Veja como enviar e-mails utilizando os componentes da paleta Indy
[*Maikel Marcelo Scheid*]

Easy

54 – Editor de Textos com RichEdit
Veja como criar seu próprio editor de textos com o componente RichEdit
[*Maikel Marcelo Scheid*]

Delphi PHP
Mini-Curso

60 – Orientação a Objetos no Delphi for PHP
Como aplicar conceitos de POO em aplicações PHP – Parte 2
[*Rodrigo Carreiro Mourão*]

Você percebeu os ícones ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Mão na Massa] Artigos que focam na resolução de problemas - e não na tecnologia. A tecnologia empregada é apenas conseqüência. O artigo parte do pressuposto que o leitor já conhece a tecnologia, mas tem dificuldade de implementá-la em uma aplicação cotidiana. Por exemplo, um artigo que ao invés de "debulhar" todos os métodos e propriedades de um "Recordset", mostre como usar o Recordset de forma inteligente. para criar um carrinho de compras.

[Delphi PHP] Artigo com foco na versão PHP do Delphi

[Engenharia de Software] Artigo dentro do contexto de Engenharia de Software

[Web] Artigos sobre ou que envolvam técnicas de desenvolvimento para WEB.

[Expert] Artigo com foco no leitor avançado

[Easy] Com foco no desenvolvedor iniciante

[Mini-Curso] Artigo que faz parte de um mini-curso

[Tutorial] Artigo no estilo tutorial passo a passo

ClubeDelphi

Ano 8 - 94ª Edição - 2008 - ISSN 1517990-7 **Impresso no Brasil**

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Adriano Santos
adrianosantos@devmedia.com.br

Equipe Editorial
Fabricio Desbessel, Maikel Scheid, Paulo Quicoli, Luciano Pimenta

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Diagramação
Adolfo Sabino
simininu@yahoo.com.br

Capa
Antonio Xavier
webdesigner@devmedia.com.br

Revisão
Gregory Monteiro
gregory@clubedelphi.net

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 3382-5025

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5025

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS. Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Apoio
ASSESPRO PARANÁ

EDITORIAL

O DataSetProvider tem uma importante função no ciclo de vida da manipulação de dados em uma aplicação Delphi. Ele é responsável por se comunicar com o engine (dbExpress por exemplo), obter os dados, empacotá-los nos chamados DataPackets e enviá-los através da memória para o ClientDataSet (em uma aplicação 2-tier) ou através da interface IAppServer em uma aplicação multicamadas (SOAP, COM+ etc.). O DataSetProvider faz muito mais do que isso, e expõe muitas de suas funcionalidades na forma de propriedades e eventos, como mostra o Adriano Santos em seu completo artigo sobre o assunto.

Falando em DataSetProvider, seu irmão ClientDataSet também exerce importante função em aplicações de banco de dados. Um dos problemas relacionados a esse componente, e que confunde muitos, é a correta manipulação de erros gerados no SGBD (Firebird por exemplo). No artigo do Rodrigo, veja como construir um descendente de ClientDataSet que automaticamente trata os principais erros do servidor, como violação de foreign-key.

Na seção Expert, o Gustavo Chaurais continua o artigo sobre Streams. Na parte final do seu artigo, veja como implementar compactação, download em múltiplos pacotes e resources em suas aplicações. Ainda nesta seção, o Marco e o Carma aprofundam um assunto muito importante, que na verdade é um desafio nos projetos de software: como mensurar o esforço e o prazo necessário para desenvolver determinada aplicação, ajudando a reduzir custos e não estropolar o cronograma.

O Maikel continua a série que demonstra a construção de uma locadora Web. E nada melhor que o ASP.NET para construir um sistema completo desse tipo. O Ricardo usa e abusa de seus conhecimentos sobre desenvolvimento com PocketStudio e continua seu excelente curso que mostra como criar uma aplicação para o sistema operacional PalmOS.

Na sessão Easy Delphi, para quem está iniciando, temos o Maikel com dois artigos: veja como utilizar os principais controles da VCL na construção de um pequeno editor de textos, bem prático. No outro artigo, mostra como utilizar os componentes da paleta Indy para criar um sistema de envio de e-mails. ideal por exemplo se você precisar enviar e-mails automaticamente da sua aplicação.

Para finalizar, o Rodrigo apresenta aquele que considero o melhor recurso da POO, o polimorfismo, tudo em PHP! Se você já domina ou conhece o assunto, sabe que ele é realmente poderoso. Que tal utilizá-lo agora em suas aplicações Web com o Delphi for PHP?

Grande abraço e sucesso com o Delphi!

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 560 Vídeo Aulas!
- 7 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 150 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/potal.asp* e utilize as informações abaixo: **Login: DVM.PL e Senha: STX200**

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Assinatura
ClubeDelphi PLUS
Mais conteúdo .NET por menos!

Informativo ClubeDelph

Portal ClubeDelphi

www.clubedelphi.net/portal

+560 vídeo aulas e 7 cursos online

Caro Leitor

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 560 vídeo aulas; ii) 7 cursos online; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 150 artigos exclusivos (que não foram publicados na revista)!;

Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).
Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou bancas, utilize o login e senha publicados na pág do editorial desta edição.
Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia

Brinde na web desta edição

1 Vídeo

Confira no portal ClubeDelphi PLUS um mini-curso sobre criação de um site com ASP.NET e SQL Server 2005 Express

http://www.devmedia.com.br/articles/listcomp.asp?txtsearch=Delphi+e+SQL+Server+2005+Express

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Últimas Vídeo-Aulas

Aprenda a desenvolver sistemas para o sistema operacional PalmOS
Acompanhe as aulas de Ricardo Boaro que falam unicamente do desenvolvimento de aplicações para PalmOS utilizando o IDE PocketStudio que é bastante semelhate ao Delphi inclusive utilizando-se de linguagem Pascal.

Curso Aplicação ASP.NET com Delphi e SQL Server 2005 Express-Parte IV a X – Testando a classe de locação de fitas
Veja nessa vídeo aula de Luciano Pimenta as principais diferenças de sintaxe para Stored Procedures e Triggers entre FireBird e SQL Server 2005.

Desenvolvendo uma aplicação para PalmOS com PocketStucio - Parte XIII a XV
Veja nessas vídeo-aulas de Ricardo Boaro, como trabalhar com aplicações PalmOS com o PocketStudio.

Mini-Curso Controle de Versão com JEDI VCS
Veja nesse mini-curso de Adriano Santos como trabalhar com esta fabulosa ferramenta para controle de versão e gerenciamento de equipes.

Como construir um WebServices com Delphi 7
Veja nessa vídeo aula de Guinther Pauli como desenvolver e trabalhar com WebServices usando o Delphi 7.

Construindo uma ferramenta de busca de arquivos Parte I e II
Veja nessas vídeos de Paulo Quicoli como desvolver uma podeorsa ferramenta para busca arquivos no disco rígido.

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Adriano Santos**
(envie as suas para falecom@adrianosantos.pro.br)

Implementando Drag and Drop na aplicação

Olá Adriano, quando digitamos um texto no Word, selecionamos e depois arrastamos para outro programa como o Bloco de Notas por exemplo, o texto é colado. Gostaria de saber se é possível fazer essa implementação em meu sistema.

André Matheus

Olá André, respondendo a sua questão: sim, é possível implementar isso em sua aplicação, mas o processo é um tanto trabalhoso. Vejamos como fazer isso.

Primeiramente devemos criar uma série de funções capazes de interagir com o mundo externo, já que possibilitaremos que o aplicativo receba o valor de um texto vindo de outro aplicativo. Para isso abra seu Delphi, salve o projeto e declare algumas funções na área *Interface* da *Unit* como segue na **Listagem 1.**

Pressione *Ctrl* + *Shift* + *C* para que o Delphi crie o cabeçalho das funções. Em seguida digite o código de cada função conforme a **Listagem 2**.

Por fim deemos alterar os eventos *OnCreate* e *OnDestroy* do formulário principal para se adaptarem às funções criadas. Veja o código na **Listagem 3**.

Adicione a *Unit ShellApi* ao *Uses* do projeto e por fim, insira um componente *Memo* na tela e execute o programa. Pra testar, abra o Word ou WordPad, digite alguma coisa, selecione e em seguida arraste-o para o *Memo* do programa criado. Note que o texto é totalmente copiado.

Listagem 1. Declaração de métodos na seção Private

```
...
private
  { Private declarations }
  function DragEnter(const dataObj: IDataObject;
      grfKeyState: Longint; pt: TPoint;
      var dwEffect: Longint): HResult; stdcall;
    function DragOver(grfKeyState: Longint;
      pt: TPoint; var dwEffect: Longint): HResult; stdcall;
    function DragLeave: HResult; stdcall;
    function Drop(const dataObj: IDataObject;
      grfKeyState: Longint; pt: TPoint; var dwEffect: Longint): HResult; stdcall;
    function _AddRef: Integer; stdcall;
    function _Release: Integer; stdcall;
public
...
```

Listagem 2. Códigos dos métodos para interagir com o mundo externo

```
function TMemoDragDropFrm.DragEnter(const dataObj:
  IDataObject; grfKeyState: Longint; pt: TPoint;
  var dwEffect: Longint): HResult;
begin
  dwEffect := DROPEFFECT_COPY;
  Result := S_OK;
end;
function TMemoDragDropFrm.DragOver(grfKeyState: Longint;
  pt: TPoint; var dwEffect: Longint): HResult;
begin
  dwEffect := DROPEFFECT_COPY;
  Result := S_OK;
end;
function TMemoDragDropFrm.DragLeave: HResult;
begin
  Result := S_OK;
end;
function TMemoDragDropFrm._AddRef: Integer;
begin
  Result := 1;
end;
function TMemoDragDropFrm._Release: Integer;
begin
  Result := 1;
end;
function TMemoDragDropFrm.Drop(const dataObj:
  IDataObject; grfKeyState: Longint; pt: TPoint;
  var dwEffect: Longint): HResult;
var
  aFmtEtc: TFORMATETC;
  aStgMed: TSTGMEDIUM;
  pData: PChar;
begin
  if (dataObj = nil) then
    raise Exception.Create('Ponteiro não á válido!');
  with aFmtEtc do
  begin
    cfFormat := CF_TEXT;
    ptd := nil;
    dwAspect := DVASPECT_CONTENT;
    lindex := -1;
    tymed := TYMED_HGLOBAL;
  end;
  OleCheck(dataObj.GetData(aFmtEtc, aStgMed));
  try
    pData := GlobalLock(aStgMed.hGlobal);
    Memo1.Text := pData;
  finally
    GlobalUnlock(aStgMed.hGlobal);
    ReleaseStgMedium(aStgMed);
  end;
  Result := S_OK;
end;
```

Mudando fontes do programa de uma só vez

Oi, gostaria de mudar todas as fontes do meu projeto automaticamente. Como posso fazer isso?

Marcelo Jibas

Olá Marcelo, uma alternativa para isso seria criar uma função que fizesse o serviço pra ti. Essa função teria que ser implementada em todos os formulários. Vejamos isso.

Crie um novo projeto no Delphi e na área *private* declare uma nova *procedure* como segue.

```
..
  private
    procedure ModificarFontes(
      AControle: TWinControl);
  public
..
```

Pressione *Ctrl* + *Shift* + *C* para que o Delphi crie o cabeçalho da função e em seguida digite o código da **Listagem 4**. No caso estamos fazendo um laço *for* nos componentes do formulário atual e utilizando os métodos da *Unit TypeInfo*, portanto declare a sua *Unit* no *Uses* do projeto. O nome da *Unit* é *TypInfo*. ●

Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback

Listagem 3. Código dos eventos OnCreate e OnDestroy

```
procedure TMemoDragDropFrm.FormCreate(Sender: TObject);
begin
  OleInitialize(nil);
  OleCheck(RegisterDragDrop(Handle, Self));
end;

procedure TMemoDragDropFrm.FormDestroy(Sender: TObject);
begin
  RevokeDragDrop(Handle);
  OleUninitialize;
end;
```

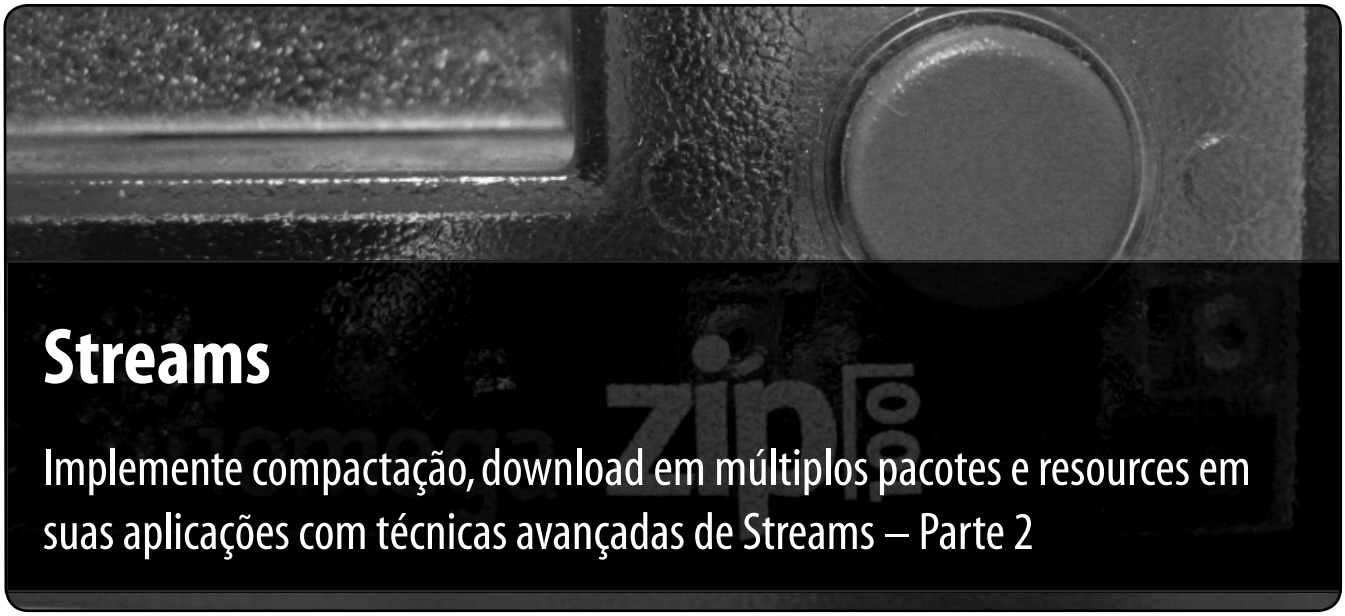
Listagem 4. Código para modificação de fontes

```
procedure TForm1.ModificarFontes(AControle:
  TWinControl);
  procedure Modificar(AControle: TControl);
  var
    f: TFont;
  begin
    if IsPublishedProp(AControle, 'Parentfont')
      and (GetOrdProp(AControle, 'Parentfont') =
        Ord(False))
      and IsPublishedProp(AControle, 'font') then
    begin
      f := TFont(GetObjectProp(AControle,
        'font', TFont));
      f.Name := 'Courier New';
    end;
  end;
var
  i: Integer;
begin
  Modificar(AControle);
  for i := 0 to AControle.ControlCount - 1 do
    if AControle.Controls[I] is TWinControl then
      ModificarFontes(TWincontrol(AControle.
        Controls[I]))
    else
      Modificar(AControle.Controls[I]);
end;
```

Para utilizar a *procedure* basta incluir alguns controles em tela e em um botão fazer a chamada a ela, com mostrado a seguir:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ModificarFontes(Self);
end;
```

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET



Streams

Implemente compactação, download em múltiplos pacotes e resources em suas aplicações com técnicas avançadas de Streams – Parte 2

Gustavo Chaurais
(gustavoc@macrovision.com)
é Borland Delphi 7 Advanced Certified, Borland Delphi 2005 for Win32 Certified, Borland Delphi 2006 for Win32 Certified e Borland Delphi Instructor Certified. Foi palestrante das três últimas edições da Borland Conference Brasil e de outros grandes eventos nacionais. Além disso, é membro da coordenadoria do GIESSC. Hoje, ministra cursos, presta consultoria e atua como Software Engineer do projeto InstallAnywhere para a empresa norte-americana Macrovision Corporation.

Na edição anterior, iniciamos a criação de uma aplicação que faz uso intensivo de streams (Edição 93), que implementa compactação, download em múltiplos pacotes e resources. Desta vez, utilizaremos um *stream* temporário para escrever o arquivo que estamos lendo. Note que estamos utilizando *GetDestStream* e *CleanDestStream* para isso (veja última listagem da edição anterior). Portanto, poderemos estender essa classe futuramente para gerar a saída em um *stream* qualquer, não só em arquivos. Seguem suas implementações (não se esqueça da diretiva *virtual* neles. Veja na **Listagem 1** a implementação das funções *GetDestStream* e *CleanDestStream*.

Seu funcionamento também é básico: descartamos do arquivo *Index* – 1 entradas para posicionar o cursor do *stream* de leitura; depois lemos o tamanho do nome, o nome e o tamanho do arquivo, também para posicionar o cursor; e, finalmente, lemos o conteúdo do arquivo através de um *CopyFrom* do *stream* temporário.

Na **Listagem 2** podemos ver a implementação de outro método importante, *ReadEntry*. Estamos apenas lendo uma entrada e descartando-a caso *WriteResult* seja false. Para finalizar, vejamos o método *Delete* descrito na **Listagem 3**.

Este é provavelmente o método mais complicado da classe. Vamos precisar de um *stream* temporário que será, na verdade, o mesmo arquivo final, só que sem o arquivo *deletado*. Tivemos de fazer isso porque, ao *deletarmos* o arquivo no mesmo *stream*, depois não conseguimos reduzir o seu *size* para o tamanho novo (menor). Esse é o comportamento padrão de um *TCompressionStream*.

O primeiro passo será então copiar para este arquivo novo todas as entradas até a de número *Index* (*ReadEntry(True)*). Depois ignoramos uma entrada (*ReadEntry(False)*) e copiamos o resto do

arquivo (*CopyEntireStream*). Em seguida, substituímos o arquivo final atual pelo novo e *repopulamos* as entradas, pois estas foram modificadas.

Pronto. Isso é tudo o que precisávamos para termos uma classe totalmente funcional com compactação para múltiplos arquivos. Agora precisamos construir uma *interface* com o usuário.

Criando um exemplo completo de compactação

No projeto atual, você deve ter ignorado o *Form* padrão, gerado pelo Delphi. Agora, volte a ele, mude seu nome para "fmCompression" e adicione um componente *TToolBar*. A este, adicione cinco *TToolButtons* ("tbNew", "tbOpen", "tbAdd", "tbRemove" e "tbExtract"). Adicione também um *TDBGrid*("dbgEntries"), um *TClientDataSet* ("cdsEntries"), um *TDataSource* ("dsEntries"), dois *TOpenDialogs*("OpenDialog" e "AddFileDialog") e um *TSaveDialog*("SaveDialog"). Opcionalmente, adicione um *TImageList* para adicionar imagens ao *TToolBar*.

Configure *dbgEntries*, apontando-o para *dsEntries* e este para o *cdsEntries*. Na propriedade *Options* de *dbgEntries*, acrescente *dgMultiSelect*. Adicione o seguinte filtro aos componentes *OpenDialog* e *SaveDialog*: "ClubeDelphi Zip File (*.cdz)|*.cdz". Seu filtro inicial deve ser configurado para "*.cdz". Dê também um título aos diálogos. Adicione a opção *ofAllowMultiSelect* à propriedade *Options* do componente *AddFileDialog*.

Abra o *Fields Editor* do componente *cdsEntries* e adicione os seguintes campos: "PATH" (*string - size*: 256), "FILE_SIZE" (*largeint*) e "ENTRY_INDEX" (*integer - visible*: *False*). Agora clique com o botão direito em *cdsEntries* e selecione *Create DataSet*.

O componente *cdsEntries* armazenará as entradas do *TCompressor*. Estamos utilizando um *TClientDataSet* com dados em memória pela facilidade em se trabalhar com os dados e com o *TDBGrid*. Opcionalmente, você pode configurar a propriedade *IndexFieldNames* do *cdsEntries* para "PATH" para ordenar as entradas por este campo. Posicione os componentes conforme a **Figura 1**.

Antes da implementação de qualquer evento, declare *FCompressor* (*TCompressor*) na seção *private* da classe *TfmCompression* e vamos à implementação do método *ReloadEntries* conforme a **Listagem 4**.

Este método é muito simples, estamos apagando os dados de *cdsEntries* e adicionando a ele todas as entradas do compressor. Além de estarmos controlando a atualização do *dbgEntries* através dos métodos *DisableControls* e *EnableControls*. Vamos implementar as operações *New* e *Open*. Para isto, manipule o evento *OnClick* do botão *tbNew* de acordo com a **Listagem 5**.

O evento *OnClick* do botão *tbOpen* deve ser implementado da mesma maneira. Porém, troque *SaveDialog* por *OpenDialog* e de *True* para *False* na construção de *FCompressor*. Agora, uma vez criado *FCompressor*, é necessário destruí-lo. Faremos isso através do evento *OnDestroy* do formulário:

```
if Assigned(FCompressor) then
  FreeAndNil(FCompressor);
```

Finalmente, vamos as outras operações. Comecemos pelo *Add*. Portanto manipule o evento do botão *tbAdd* seguindo a codificação prevista na **Listagem 6**.

Listagem 1. Métodos GetDestStream e CleanDestStream

```
function TCompressor.GetDestStream(const DestFilePath:
  string): TStream;
begin
  Result := TFileStream.Create(DestFilePath, fmCreate);
end;
procedure TCompressor.CleanDestStream(const DestStream: TStream);
begin
  DestStream.Free;
end;
```

Listagem 2. Método ReadEntry

```
procedure TCompressor.ReadEntry(WriteResult: Boolean);
var
  Buffer: TBuffer;
  NameLength: Integer;
begin
  FStreamRead.Read(Buffer, 4);
  if WriteResult then
    FStreamWrite.Write(Buffer, 4);
  NameLength := BinToInt(CopyFromBuffer(Buffer, 4));
  FStreamRead.Read(Buffer, NameLength);
  if WriteResult then
    FStreamWrite.Write(Buffer, NameLength);
    FStreamRead.Read(Buffer, 8);
  if WriteResult then
    FStreamWrite.Write(Buffer, 8);
  ReadFile(BinToInt64(CopyFromBuffer(Buffer, 8)), WriteResult);
end;
```

Listagem 3. Método Delete

```
procedure TCompressor.Delete(Index: Integer);
var
  i: Integer;
  ATempStream: TFileStream;
begin
  RestartStream(fmOpenRead);
  try
    ATempStream := TFileStream.Create(FFilePath + '.tmp', fmCreate);
    try
      FStreamWrite := TCompressionStream.Create(clMax, ATempStream);
      FStreamRead := TDecompressionStream.Create(FInternalStream);
      for i := 0 to Index - 1 do
        ReadEntry(True);
      ReadEntry(False);
      CopyEntireStream;
    finally
      CloseStreams;
      ATempStream.Free;
    end;
  finally
    CloseStreams;
  end;
  DeleteFile(FFilePath);
  RenameFile(FFilePath + '.tmp', FFilePath);
  RestartStream(fmOpenRead);
  FStreamRead := TDecompressionStream.Create(FInternalStream);
  try
    PopulateEntries;
  finally
    CloseStreams;
  end;
end;
```

Listagem 4. Código do método ReloadEntries

```
procedure TfmCompression.ReloadEntries;
var
  i: Integer;
  Entry: TCompressionEntry;
begin
  cdsEntries.DisableControls;
  cdsEntries.EmptyDataSet;
  try
    for i := 0 to FCompressor.EntriesCount - 1 do
    begin
      Entry := FCompressor.Entries[i];

      cdsEntries.Append;
      cdsEntriesPATH.AsString := Entry.Path;
      cdsEntriesFILE_SIZE.AsLargeInt := Entry.
        FileSize;
      cdsEntriesENTRY_INDEX.AsInteger := i;
      cdsEntries.Post;
    end;
  finally
    cdsEntries.EnableControls;
  end;
end;
```

Listagem 5. Código do botão Novo

```
procedure TfmCompression.tbNewClick(Sender: TObject);
var
  AFileName: string;
begin
  if SaveDialog.Execute then
  begin
    AFileName := SaveDialog.FileName;
    if Assigned(FCompressor) then
      FCompressor.Free;
    FCompressor := TCompressor.Create(AFileName,True);
    Caption := ExtractFileName(AFileName);
    ReloadEntries;
  end;
end;
```

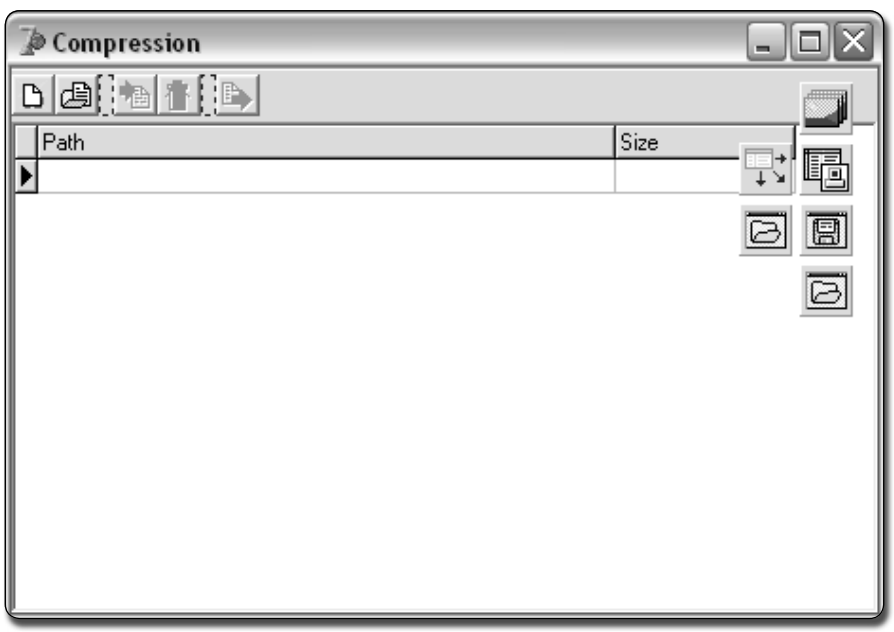


Figura 1. Interface gráfica do projeto Compression

Mais um método bastante simples, no qual passamos ao *FCompressor* todos os arquivos a serem adicionados. Para a operação de *Extract*, implemente o evento *OnClick* do botão *tbExtract* (**Listagem 7**)

A função *SelectDirectory* não é muito conhecida. Esta serve para que o usuário possa selecionar um diretório, como um diálogo qualquer. Passamos o *Caption*, o diretório inicial e para onde enviar o diretório selecionado. Então, para cada linha selecionada no *dbgEntries* estamos chamando o método *Extract* do compressor passando o *Index* da entrada, armazenado no método *ReloadEntries*.

Novamente, o caso mais complicado. Manipule o evento *OnClick* do botão *tbDelete* para usando o código da **Listagem 8**.

Para entender o código anterior, precisamos pensar em uma operação na qual estão sendo removidos vários registros de uma coleção. Caso removamos um registro e, posteriormente, tentemos remover outro de índice maior, o segundo registro estará sendo removido de maneira incorreta. Portanto, vamos fazer a operação pegando do maior índice a ser removido para o menor.

Por facilidade, vamos utilizar um *TStringList* para armazenar os índices. A primeira tarefa é então alimentar o *TStringList* com os índices das entradas a serem removidas. A segunda compreende procurar o próximo índice mais alto da lista, guardá-lo e salvar sua posição na coleção de índices. Após isso, *deletamos* a entrada e a excluímos da coleção. Quando todos os registros forem removidos, simplesmente carregamos todas as entradas novamente.

Pronto. Nossa versão simplificada do *WinZip* para trabalhar com arquivos *.cdz* está pronta para ser utilizada. Você pode agora tentar adicionar outros recursos como criptografia ao seu arquivo compactado. Além disso, como criamos uma classe, você pode embutir essa ferramenta em sua aplicação para transportar arquivos com mais facilidade.

Trabalhando com Resources

Todo programa Windows de interface gráfica apresenta uma série de recursos que não são apenas código compilado, como:

ícones, cursores, sons, imagens, dentre outros. O próprio Windows nos permite inserir, em nossos executáveis, qualquer tipo de arquivo. Estes podem ser compilados em arquivos de recursos (*.RES*) e então embutidos em uma aplicação.

Esta é a receita de bolo para a utilização de arquivos de recursos:

1. Criamos um arquivo *.RC*, referenciando os arquivos que irão constar no *.RES*;
2. Compilamos o arquivo *.RC* para *.RES*;
3. No Delphi, utilizamos a diretiva *{$R Arquivo.RES}* para importar o arquivo de recursos;
4. Fazemos uso da classe *TResourceStream* para a leitura dos arquivos.

Nosso primeiro exemplo é um clássico da customização dos recursos de uma aplicação. Certamente, se nunca utilizou, irá utilizá-lo um dia em sua aplicação.

É muito comum fazermos uso de bibliotecas externas ao nosso programa. Contudo, o fato de termos de distribuí-las junto ao nosso executável nos incomoda bastante. Portanto, vamos incluí-las em nossa aplicação. Deste modo, poderíamos embutir também o executável final e extrair todos eles antes da execução propriamente dita.

Crie (com o próprio bloco de notas) um arquivo com a extensão *.RC* no disco (por exemplo: *Recursos.RC* na pasta da aplicação). Seu conteúdo ficaria parecido com o seguinte:

```
DLL DLLFILE caminho_para_biblioteca.dll
```

A primeira coluna contém a chave que identificará o arquivo como recurso e poderá ser representada por qualquer palavra. A segunda coluna contém o tipo de arquivo. Os mais comuns são: *BITMAP*, *ICON*, *CURSOR*, *JPEG*, *WAVE*, *TEXT* e *RCDATA* (genérico). A terceira coluna contém o caminho (absoluto ou relativo) para o arquivo. Lembre-se de que inúmeras linhas (entradas) podem ser adicionadas ao arquivo *.RC*. Aponte para qualquer biblioteca em seu sistema para testar.

Com o fonte *.RC* pronto, devemos compilá-lo para um *.RES*. Para isso, você pode utilizar o programa *BRCC32.exe*, na pasta *bin* do *Delphi*. Abra uma linha de comando e digite:

Listagem 6. Código do botão Add

```
procedure TfmCompression.tbAddClick(Sender: TObject);
var
  i: Integer;
  Entry: TCompressionEntry;
begin
  cdsEntries.DisableControls;
  try
    if AddFileDialog.Execute then
    begin
      for i := 0 to AddFileDialog.Files.Count - 1 do
      begin
        Entry := FCompressor.Add(AddFileDialog.
          Files[i]);
        cdsEntries.Append;
        cdsEntriesPATH.AsString := Entry.Path;
        cdsEntriesFILE_SIZE.AsLargeInt := Entry.
          FileSize;
        cdsEntriesENTRY_INDEX.AsInteger := FCompressor.
          EntriesCount - 1;
        cdsEntries.Post;
      end;
    end;
  finally
    cdsEntries.EnableControls;
  end;
end;
```

Listagem 7. Código do botão de extração

```
procedure TfmCompression.tbExtractClick(Sender: TObject);
var
  ExtractTo: string;
  i: Integer;
begin
  if SelectDirectory('Extract to', 'c:\', ExtractTo)
  then
  begin
    cdsEntries.DisableControls;
    try
      for i := 0 to dbgEntries.SelectedRows.Count-1 do
      begin
        cdsEntries.GotoBookmark(Pointer(dbgEntries.
          SelectedRows.Items[i]));
        FCompressor.Extract(cdsEntriesENTRY_INDEX.
          AsInteger, ExtractTo);
      end;
    finally
      cdsEntries.EnableControls;
    end;
  end;
end;
```

Listagem 8. Código para remoção de arquivo

```
procedure TfmCompression.tbRemoveClick(Sender: TObject);
var
  i: Integer;
  ToRemove: TStrings;
  EntryIndexToRemove, iToRemove: Integer;
begin
  cdsEntries.DisableControls;
  ToRemove := TStringList.Create;
  try
    for i := 0 to dbgEntries.SelectedRows.Count-1 do
    begin
      cdsEntries.GotoBookmark(Pointer(dbgEntries.
        SelectedRows.Items[i]));
      ToRemove.Add(cdsEntriesENTRY_INDEX.AsString);
    end;
    while ToRemove.Count > 0 do
    begin
      EntryIndexToRemove := -1;
      iToRemove := -1;
      for i := 0 to ToRemove.Count - 1 do
      begin
        if StrToInt(ToRemove[i])>EntryIndexToRemove
        then
        begin
          iToRemove := i;
          EntryIndexToRemove := StrToInt(ToRemove[i]);
        end;
      end;
      FCompressor.Delete(EntryIndexToRemove);
      ToRemove.Delete(iToRemove);
    end;
  finally
    cdsEntries.EnableControls;
    ToRemove.Free;
  end;
  ReloadEntries;
end;
```

```
[caminho para a pasta bin]\brcc32.exe
[arquivo .RC]
```

Na pasta atual será gerado um arquivo *.RES* com o mesmo nome de seu fonte *.RC*. Crie um novo projeto e salve-o como "ResourceDLL.dpr". Na *unit* do formulário principal, digite:

```
{$R caminho_para_o_arquivo_de_recurso_
compilado.RES}
```

Adicione um botão ao formulário e implemente seu evento *OnClick* igual a **Listagem 9.** Inicie o programa e clique no botão. A DLL será extraída para a mesma pasta, com o nome *extracted.dll*.

Relembrando o conceito de *streams* do início do artigo, finalmente fechamos as principais classes que o assunto abrange. O *TResourceStream* é muito parecido com o *TMemoryStream* e necessita de três parâmetros para sua construção. O primeiro é o *HModule*, ou seja, o *handle* do módulo retornado no carregamento de uma aplicação/DLL. No caso, o arquivo de recursos foi compilado junto à aplicação corrente, portanto, utilizaremos *HInstance*. No caso de querermos carregar recursos de outras aplicações passaríamos o *HModule* da aplicação desejada. O segundo parâmetro é a chave que identifica o recurso, informada no *.RC*. E o terceiro é o tipo de recurso, também informado no *.RC*.

A classe *TResourceStream*, bem como o *TMemoryStream*, herdam de *TCustomMemoryStream*. Esta última introduz dois métodos especiais: *SaveToFile* e *SaveToStream*. Isso é muito útil porque não precisamos de um *TFileStream* para salvar seu conteúdo. O método *SaveToFile* já faz isso para nós. E esta é a operação presente em nosso exemplo. Estamos simplesmente criando o *stream* e chamando seu método *SaveToFile*. Muito simples, não?

Como dica, podemos utilizar o plug-in *Simple Resource Editor* (**Figura 2**), de minha autoria, para a manipulação dos arquivos *.RC*. Com ele você pode facilmente adicionar novos arquivos (buscando-os por diálogos especiais), compilar para arquivos *.RES*, dentre outras funcionalidades bastante úteis. O assistente lhe ajuda inclusive a, rapidamente, selecionar o tipo de recurso a ser adicionado. Seu download pode ser feito através do *CodeCentral* da *CodeGear* (*cc.codegear.com)*, procurando-se pelo autor *Gustavo Chaurais*. Seu código fonte está incluso e é muito interessante para estudo.

Listagem 9. Código do botão de extração dos recursos

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Stream: TResourceStream;
const
  EXTRACTED_DLL_NAME = 'extracted.dll';
begin
  Stream := TResourceStream.Create(HInstance, 'DLL',
    'DLLFILE');
  try
    Stream.SaveToFile(EXTRACTED_DLL_NAME);
    MessageDlg('Arquivo extraído para: ' +
      IncludeTrailingPathDelimiter(ExtractFilePath(
      Application.ExeName)) + EXTRACTED_DLL_NAME,
      mtInformation, [mbOk], 0);
  finally
    FreeAndNil(Stream);
  end;
end;
```

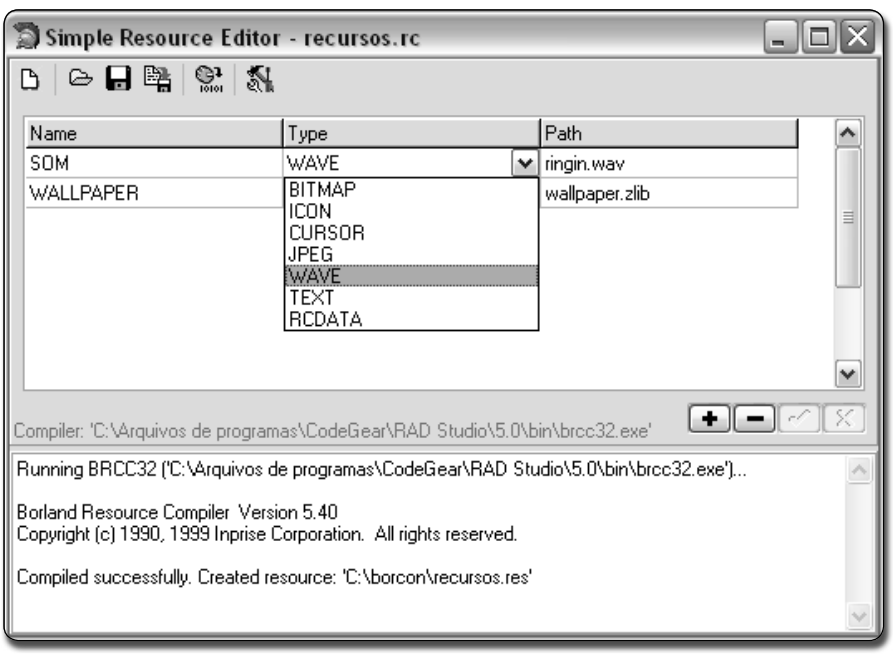


Figura 2. Simple Resource Editor

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Adriano Santos que mostra como trabalhar com o plug-in Simple Resource Editor.

www.devmedia.com.br/articles/viewcomp.asp?comp=5476&hl=

Vamos agora misturar um pouco as coisas. Construiremos um exemplo que utilizará um arquivo comprimido através do exemplo *Compression*.

Crie um novo projeto e salve-o como *ResourceZip.dpr*. Adicione a ele uma nova *Unit* e salve-a como *ResCompressor.pas*. Adicione também a *Unit Compressor.pas*, construída anteriormente.

Na seção *Uses* de *ResCompressor*, insira: *Compressor, Classes, SysUtils* e *Windows*. Agora, declare uma classe *TResCompressor*, estendendo *TCompressor*. Declare e implemente o seguinte método (com a diretiva *reintroduce*):

```
constructor TResCompressor.Create(const
ResName:
  string);
begin
  FResName := ResName;
  inherited Create('', False);
end;
```

E declare *FResName* na seção *private* como sendo do tipo *string*. Declare também e implemente o seguinte método:

```
procedure TResCompressor.
ExtractToStream(Index:
  Integer; const ExtractTo: TStream);
begin
  FDestStream := ExtractTo;
  inherited Extract(Index, '');
end;
```

Adicione também *FDestStream* à seção *private* da classe. Este método será utilizado para extrairmos um *resource* para um *stream* qualquer.

Você se lembra dos três métodos virtuais da classe *TCompressor*? Pois, tenha certeza de que não estão na seção *private* da classe *TCompressor* (devem estar preferencialmente em *protected*) e redeclare-os na classe *TResCompressor* com a diretiva *override*. O atributo de *TCompressor*, *FInternalStream*, também deve ser colocado preferencialmente na sua seção *protected*.

O método *RestartStream* deve ser utilizado para a inicialização do *stream* interno. Portanto, o inicializaremos como um *TResourceStream* conforme segue:

```
procedure TResCompressor.RestartStream(
  Mode: Word);
begin
  FInternalStream := TResourceStream.Create(
    HInstance, FResName, RT_RCDATA);
end;
```

O método *GetDestStream* é utilizado para a inicialização de uma *string* onde será gravado o arquivo extraído. Neste caso, quando for chamado o método *ExtractToStream*, o *stream* desejado será gravado localmente em *FDestStream*. Podemos então simplesmente retorná-lo. Veja:

```
function TResCompressor.GetDestStream(const
  DestFilePath: string): TStream;
begin
  Result := FDestStream;
end;
```

E, uma vez que o *stream* veio de fora para o *ExtractToStream*, não é nossa responsabilidade limpá-lo. Provavelmente, quem chamou o método vai querer ler o que foi extraído. Portanto, o método *CleanDestStream* ficará vazio. Note a seguir:

```
procedure TResCompressor.
CleanDestStream(const
  DestStream: TStream);
begin
end;
```

Para fechar esta classe, você pode, opcionalmente, disparar uma exceção caso sejam chamados os métodos *Add*, *Delete* e *Extract*. Para tanto, declare-os como *virtual* na classe *TCompressor* e *override* na classe *TResCompressor*.

Agora, vamos construir nosso arquivo de recursos. Abra o programa *Compression* e crie um arquivo contendo algumas figuras do tipo *jpg*. Crie um arquivo *.RC* e adicione a ele a seguinte linha:

```
IMAGESZIP RCDATA caminho_para_arquivo_
compactado.cdz
```

Compile o *.RC* para um *.RES* e o referencie no formulário principal, através da diretiva *{$R Arquivo.RES}*. Adicione ao formulário principal da aplicação um *TPaintBox* (aba *System*). Na sua *Unit*, insira *jpeg* na seção *Uses* e manipule o evento *OnCreate* digitando o código da **Listagem 10**.

O que este método está fazendo se refere à criação de um *TResCompressor*, o qual irá carregar o *.RES*, contendo o arquivo das figuras compactado. Após sua abertura, podemos chamar seu método *ExtractToStream* passando um índice aleatório. Finalmente, carregamos à figura em *FJpeg* (declare-a na classe do formulário). Utilizamos um *TMemoryStream* para carregá-la. Agora, no evento *OnPaint* do *TPaintBox*, implemente o código seguinte.

```
procedure TForm1.PaintBox1Paint(Sender:
TObject);
begin
  if Assigned(FJpeg) then
    PaintBox1.Canvas.Draw(0, 0, FJpeg);
end;
```

Pronto. Agora, toda vez que você entrar no programa, uma figura aleatória de sua lista será carregada no *TPaintBox*. Você pode utilizar isso no *splash screen* de sua aplicação, por exemplo.

E, se você ainda não cansou de exemplos, vamos finalizar com um muito pouco conhecido: como alterar *resources* de outras aplicações.

Ainda no exemplo anterior, adicione a seguinte linha na seção *Interface* da *Unit* do formulário:

```
resourcestring
  TITLE = 'not changed';
```

Resource Strings são como constantes. No entanto, podemos modificar seu conteúdo utilizando programas que alterem *resources* como o *Resource Hacker* ou o *Workshop Editor*. Essas são muito utilizadas em programas que desejamos internacionalizar, pois podemos alterar seu conteúdo para outra língua sem ter de *recompilar* a aplicação toda. Adicione como primeira linha do manipulador do evento *OnCreate* do formulário o seguinte:

```
Caption := TITLE;
```

Crie um novo projeto e salve-o como "ResourceUpdate.dpr". No formulário principal, adicione: três *TLabel*, três *TEdit*("edProgram", "edResKey" e "edValue"), três *TButton*("btList", "btRCDATA" e "btString") e um *TMemo*("mmList"). Posicione os componentes e configure seus *Captions* conforme a **Figura 3**.

Vamos iniciar pela listagem dos recursos. Implemente o tratamento para o evento *OnClick* do *btList* observando o código da **Listagem 11**.

Estamos, primeiramente, utilizando a função *LoadLibraryEx* para carregar

Listagem 10. Código OnCreate do formulário

```
procedure TForm1.FormCreate(Sender: TObject);
var
  AResCompressor: TResCompressor;
  AIndex: Integer;
  AMemoryStream: TStream;
begin
  AResCompressor := TResCompressor.Create('IMAGESZIP');
  try
    AMemoryStream := TMemoryStream.Create;
    try
      Randomize;
      AIndex := RandomRange(0, AResCompressor.EntriesCount);
      AResCompressor.ExtractToStream(AIndex, AMemoryStream);

      AMemoryStream.Position := 0;
      FJpeg := TJpegImage.Create;
      FJpeg.LoadFromStream(AMemoryStream);
    finally
      AMemoryStream.Free;
    end;
  finally
    AResCompressor.Free;
  end;
end;
```

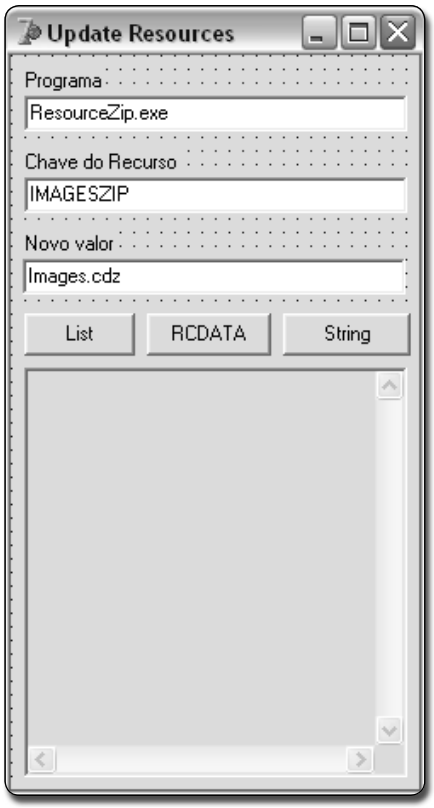


Figura 3. Interface do exemplo ResourceUpdate

o programa. Esta função pode ser executada tanto para bibliotecas quanto executáveis e retorna o *handle* do módulo. Após isso, chamamos o método *EnumResourceNames*, uma vez para o tipo *RCDATA* e uma vez para o tipo *STRING*. Passamos também um ponteiro para a função que será chamada para cada recurso (*@EnumResNamesProc*) e um ponteiro para a lista de *strings* do *mmList*.

Feito isso, declare (logo acima do método recém implementado, sem ser da classe *TForm1*) a função *EnumResNmaesProc* (**Listagem 12**).

Quando o recurso for do tipo *string*, na verdade, ele será um *StringTable*. Essas tabelas são a maneira que *strings* são adicionadas aos arquivos de recurso. Em cada uma, teremos até 16 *strings*. Cada *string* é identificado por uma chave própria, que é calculada da seguinte forma: ([NOME DO RECURSO] – 1) * 16. Posteriormente, podemos utilizar a função *LoadString* passando esta chave para lermos cada *string*.

Listagem 11. Código do evento OnClick do botão btList

```
procedure TForm1.btListClick(Sender: TObject);
var
  AHandle: THandle;
begin
  mmList.Clear;
  AHandle := LoadLibraryEx(PChar(edProgram.Text), 0, LOAD_LIBRARY_AS_DATAFILE);
  try
    mmList.Lines.Add('RCDATA RESOURCES');
    EnumResourceNames(AHandle, RT_RCDATA, @EnumResNamesProc, Integer(mmList.Lines));
    mmList.Lines.Add('');
    mmList.Lines.Add('STRING TABLES');
    EnumResourceNames(AHandle, RT_STRING, @EnumResNamesProc, Integer(mmList.Lines));
  finally
    FreeLibrary(AHandle);
  end;
end;
```

Listagem 12. Função EnumResNamesProc

```
function EnumResNamesProc(Module: HMODULE; ResType, ResName: PChar;
  Strings: TStrings): Boolean; stdcall;
var
  InitialString, i: Integer;
  Buffer: array[0..1023] of Char;
begin
  if ResType = RT_STRING then
  begin
    InitialString:=(LoWord(Cardinal(ResName))-1) * 16;
    for i := 0 to 15 do
    begin
      if LoadString(Module,InitialString+i,
        Buffer,1024) <> 0 then
        Strings.Add('> ' + IntToStr(InitialString + i)
        + ' - ' + string(Buffer));
    end;
  end
  else if ResType = RT_RCDATA then
  begin
    Strings.Add('> ' + ResName);
  end;
  Result := True;
end;
```

O método implementado, conforme comentado, será chamado para cada recurso encontrado. Se o recurso for um *RCDATA*, apenas adicionamos na lista o nome do recurso. Caso seja um *String Table*, varremos a tabela lendo cada *string* através do método *LoadString*. Para pegarmos o nome de recurso da tabela, utilizamos a função *LoWord*, pois, nos interessa apenas os *bytes* menos significativos.

A lista já está pronta para ser preenchida. Você verá primeiro os recursos *RCDATA* do aplicativo e, depois, os recursos do tipo *String Table*. No segundo caso, aparecerá também a chave identificadora de cada *string*.

Agora, declare e implemente o método *UpdateResourceInProgram* (**Listagem 13**).

Este é o método que será utilizado para a modificação efetiva dos *resources*. Ele consiste em chamarmos *BeginUpdateResource* para iniciar a alteração, o qual nos retorna o *Handle* do programa. Se a alteração for do tipo *RCDATA*, iniciamos um *stream* para a leitura do arquivo que irá substituir o recurso atual; reservamos memória para todo este valor (note que o arquivo será totalmente carregado em memória); lemos o arquivo no *buffer* e, por fim, chamamos *UpdateResource*. Este é o método que irá modificar o recurso e, para ele, devemos passar: o *Handle* da aplicação a modificar, o tipo de recurso, o nome do recurso (é interessante utilizar a função *MakeIntResource* para a formatação do valor), a linguagem do recurso (0 = Neutral), o *buffer* com os *bytes* e o tamanho a ser gravado.

Caso estejamos interessados na alteração de uma *string*, chamamos o método a ser implementado *GetStringTable* e gravamos, agora, toda a *String Table* novamente na posição correta (retornada por *GetResNameOfStrId*. Para aplicar as atualizações, chamamos *EndUpdateResource*.

Finalmente, implemente os métodos relacionados à construção do *String Table* conforme a **Listagem 14**.

Para a montagem do *String Table*, é necessário lermos todas as *strings* novamente, adicionando-as na tabela e alterando somente a *string* desejada. Perceba que as *strings* estão presentes na tabela da seguinte forma: [LENG-

Listagem 13. Método UpdateResourceInProgram

```
procedure TForm1.UpdateResourceInProgram(const
  ResType: PChar);
var
  TempStream: TStream;
  ResHandle: THandle;
  Buffer: PChar;
  StringTable: WideString;
begin
  ResHandle := BeginUpdateResource(PChar(
    edProgram.Text), False);
  try
    if ResType = RT_RCDATA then
    begin
      TempStream := TFileStream.Create(
        edValue.Text, fmOpenRead);
      try
        GetMem(Buffer, TempStream.Size);
        TempStream.Read(Buffer^, TempStream.Size);
        UpdateResource(ResHandle, RT_RCDATA,
          MakeIntResource(edResKey.Text), 0,
          Buffer, TempStream.Size);
      finally
        TempStream.Free;
      end;

      FreeMem(Buffer);
    end
    else if ResType = RT_STRING then
    begin
      StringTable := GetStringTable(StrToInt(
        edResKey.Text), edValue.Text);
      UpdateResource(ResHandle, RT_STRING,
        MakeIntResource(GetResNameOfStrId(StrToInt(
        edResKey.Text))), 0, PWideChar(StringTable),
        Length(StringTable));
    end;
  finally
    EndUpdateResource(ResHandle, False);
  end;
end;
```

Listagem 14. Implementação dos métodos GetStringTable e GetResNameOfStrId

```
function TForm1.GetStringTable(StrId: Integer; const
  NewValue: string): WideString;
var
  ResName: Integer;
  Offset: Integer;
  StartPos: Integer;
  i: Integer;
  AHandle: THandle;
  Buffer: array[0..1023] of Char;
  StrRead: string;
begin
  ResName := GetResNameOfStrId(StrId);
  Offset := StrId mod 16;
  StartPos := (ResName - 1) * 16;
  Result := '';
  AHandle := LoadLibraryEx(PChar(edProgram.Text), 0,
    LOAD_LIBRARY_AS_DATAFILE);
  try
    for i := 0 to 15 do
    begin
      if i = Offset then
      begin
        Result := Result + Char(Length(NewValue)) +
          NewValue;
      end
      else
      begin
        if LoadString(AHandle, StartPos + i, Buffer,
          1024) <> 0 then
        begin
          StrRead := string(Buffer);
          Result := Result + Char(Length(StrRead)) +
            StrRead;
        end;
      end;
    end;
    Result:= Result + StringOfChar(#0,Length(Result));
  finally
    FreeLibrary(AHandle);
  end;
end;
function TForm1.GetResNameOfStrId(StrId:
  Integer): Integer;
begin
  Result := (StrId div 16) + 1;
end;
```

TH][STRING]. O cálculo da posição inicial é feito com base no nome do recurso correspondente à *String Table* em questão. Já o *Offset* representa a posição da *string* a ser alterada na tabela. Para finalizar, devemos gerar o mesmo número de caracteres nulos correspondentes ao tamanho gerado. Assim, fechamos a construção de um *String Table* completo. Já o método *GetResNameOfStrId* é um simples cálculo para se saber o *resource name* do *String Table* com base no *id* de um *string*.

Teste seu programa apontando para o executável anteriormente criado *ResourceZip.exe* e clicando em *btList*. Você verá a lista com os recursos do programa. Para alterar um *string*, coloque no segundo *TEdit* o *id* da *string* (copie da lista) e dê um novo valor. Por exemplo, procure por "not changed", correspondente ao *resourcestring* anteriormente adicionado, mude seu valor e execute o programa. Você verá que a *string* foi modificada com sucesso, através da barra de títulos. Gere também outro arquivo *.cdz* com diferentes figuras *jpeg* e faça uma modificação apontando para o nome do recurso (*IMAGESZIP*) e para o caminho ao arquivo *.cdz*. Abra o programa e você verá as imagens novas.

Conclusão

Por mais que achemos que sabemos tudo, sempre há algo para aprender. *Streams* não são um conteúdo difícil, porém, necessitam ser exercitados. Além disso, aumentamos um pouco mais nosso "arsenal" de possibilidades, com a utilização de poderosos compactadores de arquivos e arquivos de recurso bastante flexíveis. Agora, é só esperar a oportunidade certa para aplicar o que foi explicado em um cenário real. ●

Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Análise de Pontos de Função

Saiba mensurar o tamanho de seu software e estimar o tempo que levará para ficar pronto

Carmo Crêdiney de Melo
(carmo@jfnet.com.br)
Possui experiência desde 1990 em Tecnologia da Informação. Bacharel em Sistemas de Informação pela Faculdade Metodista Granbery (FMG). Acumula sete anos de estudos acadêmicos na área de tecnologia e na área de Gestão de Projetos. Possui capacitação em Rational Unified Process (RUP), Arquitetura J2EE, Linux, Gestão de Projeto de Tecnologia e Análise de Pontos de Função.

Marco Antônio Pereira Araújo
(maraujo@granbery.edu.br)
É Professor do Curso de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery, Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Matemática com Habilitação em Informática pela UFJF, Analista de Sistemas da Prefeitura de Juiz de Fora.

Um dos grandes desafios atualmente em projetos de software é garantir que estimativas de tamanho, custo e cronograma sejam realistas, tanto em projetos de desenvolvimento quanto de manutenção. Entretanto, as crescentes necessidades dos usuários, a alta complexidade e baixo entendimento do domínio das aplicações a serem construídas, aliadas às constantes mudanças tecnológicas, promovem um alto grau de incerteza nas estimativas para o cumprimento das metas estabelecidas.

Como resultado, os desenvolvedores têm enfrentado problemas de subestimativas, culminando em atrasos no cronograma e descumprimento de orçamentos. Por outro lado, entende-se que os usuários também precisam de respostas rápidas em relação ao tempo e custos envolvidos em projetos de software. No sentido de obter estimativas mais confiáveis, necessita-se lançar mão de técnicas específicas para esse fim, como a Análise de Pontos de Função (do inglês FPA - *Function Point Analysis*).

Dessa forma, pode-se estimar o tamanho das funcionalidades de um sistema e, em função disso, definir o tempo e recursos para o projeto em questão com base na produtividade da equipe ou na taxa de entrega de projetos semelhantes. Essas estimativas definidas no início do projeto são determinantes para planejar as suas iterações, prever o custo e elaborar o orçamento detalhado do projeto, minimizando os riscos que o projeto termine fora do prazo ou do orçamento previsto.

Assim, o objetivo deste artigo é descrever como obter o tamanho de sistemas através da técnica de contagem de Pontos de Função, através de um protótipo de interface de uma aplicação de exemplo. Não se tem a pretensão de apresentar todas as regras de FPA, mas introduzir os conceitos básicos para a contagem em projetos de desenvolvimento.

O estudo de caso apresentado neste artigo é relativo à contagem de Pontos de Função de um fragmento de um sistema de Controle Acadêmico.

Introdução à análise de Pontos de Função

A FPA foi desenvolvida em meados da década de 70 na tentativa de minimizar as dificuldades associadas à medição de tamanho de software através de linhas de código-fonte, além de prover um mecanismo que pudesse prever o esforço associado ao desenvolvimento de software.

Em 1984, uma versão mais refinada foi lançada e, posteriormente, com o aumento da utilização da FPA, tornou-se necessário definir um guia que interpretasse as regras originais para novos ambientes. Devido a essa necessidade, em 1986 foi criado o *International Function Point Users Group* (IFPUG). No Brasil, esse grupo é representado pelo BFPUG (*Brazilian Function Point Users Group*).

O cálculo de Pontos de Função (PF) segue um conjunto de diretrizes descritas no manual de Práticas de Contagem de Pontos de Função do IFPUG. Entretanto, divergências de como aplicar essas regras fazem com que as contagens nem sempre resultem num mesmo valor.

No sentido de criar um método mais rigoroso nesse processo de contagem, surgiu a norma ISO/IEC 14143, que estabelece uma série de padrões para contagem funcional. Em 2002, a versão 4.1 do manual do IFPUG foi aprovada como aderente a essa norma.

Dessa forma, a FPA permite uma contagem indicativa do tamanho do projeto no início do seu desenvolvimento, sem conhecer detalhes de modelos de dados ou de classes. Posteriormente, na fase de construção, essa contagem representa uma estimativa com maior precisão da complexidade das funções e, ao término da construção do software, na etapa de transição, é realizada uma contagem detalhada, obtida a partir do grau de complexidade das funções levantadas no processo funcional, modelo de dados ou modelo de classes, descrição de telas e relatórios.

Contagem de Pontos de Função

A contagem de pontos de função é realizada através de um conjunto de passos previamente determinados. Para facilitar o entendimento desse processo, cada um desses passos será apresentado e aplicado a um estudo de caso que demonstrará como a contagem pode ser feita.

Nesse sentido, o artigo conduzirá um estudo de caso relacionado ao cadastro de cursos e alunos de uma faculdade. O cadastramento de cursos é iniciado pelo usuário e apresenta uma lista dos cursos já cadastrados, permitindo pesquisa a partir da descrição do curso e oferecendo as opções de cadastro relativas à inclusão, alteração, exclusão e consulta (**Figura 1**).

A partir dessa janela, o sistema deve possibilitar a manutenção no cadastro de cursos contendo código e descrição do mesmo, além do tipo do curso, que pode ser de graduação ou pós-graduação. Para cursos de graduação informa-se também o número de períodos, enquanto que, para cursos de pós-graduação, informa-se a carga horária (**Figura 2**).

Ao final de qualquer dos processos de manutenção, o sistema deve emitir uma mensagem de confirmação ao usuário.

Após o cadastramento de cursos, podemos matricular os alunos. Da mesma maneira, existe uma janela de pesquisa de alunos, praticamente igual à de pesquisa de cursos, exceto que oferece ainda

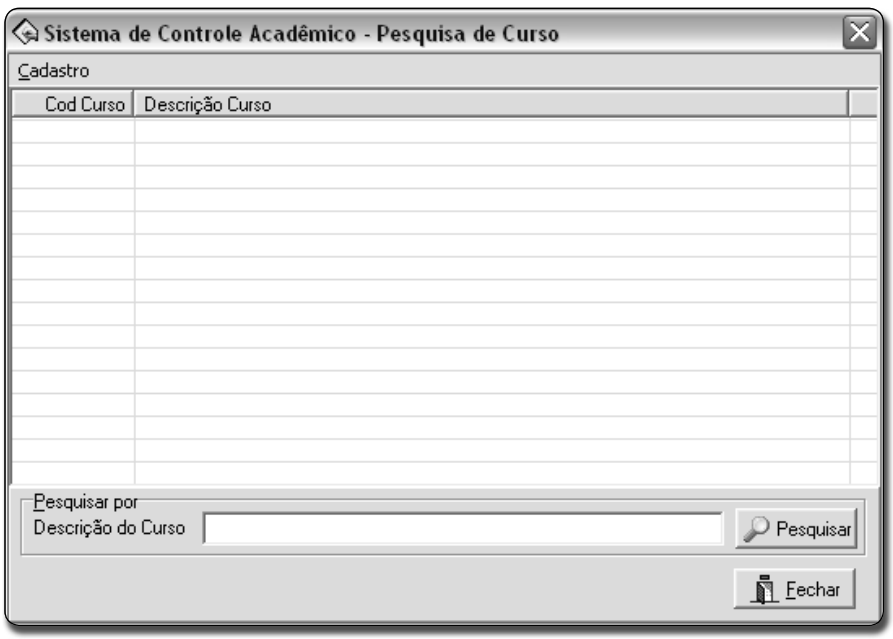


Figura 1. Janela de pesquisa de cursos

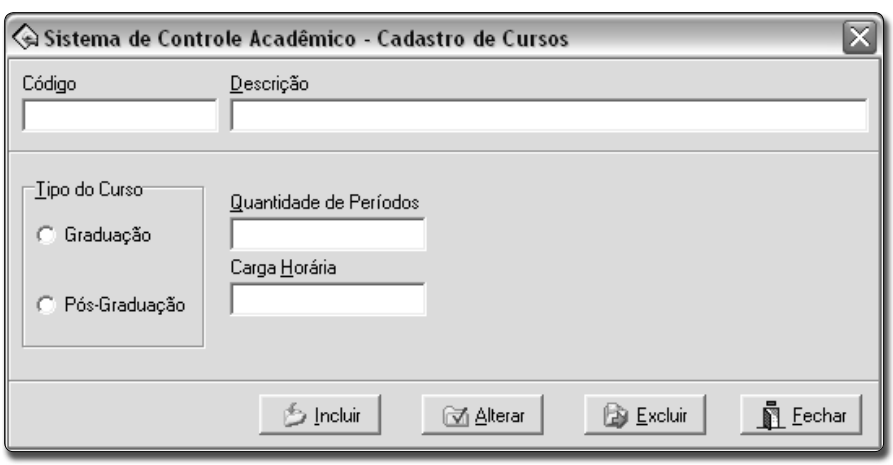


Figura 2. Janela de cadastro de cursos

uma possibilidade de filtrar os alunos por um curso específico (**Figura 3**).

O cadastramento de alunos, também semelhante ao de cursos, possui matrícula, nome, dados pessoais (**Figura 4**), dados de endereço (**Figura 5**) e dados de documentação (**Figura 6**). Após o cadastramento de um aluno, o sistema deverá exibir para o usuário o valor da mensalidade a ser pago pelo aluno.

Para o cálculo da mensalidade, um valor base é utilizado para cada tipo de curso e considera ainda o número de períodos para cursos de graduação e a carga horária para cursos de pós-graduação. Os valores base das mensalidades são mantidos por um sistema financeiro e estão fora dos limites dessa aplicação, sendo consultados por ela.

Para a construção dessas funcionalidades, será primeiramente determinado o tamanho da aplicação que será desenvolvida, de forma a orientar as estimativas de prazo e custo. Assim, para estimar o tamanho do software de acordo com a FPA, segue-se um procedimento de contagem, que está representado no esquema da **Figura 7**.

A seguir, cada uma dessas atividades será descrita e aplicada ao estudo de caso apresentado.

1) Tipo de Contagem: existem três tipos de contagem de pontos de função. A diferença no procedimento adotado entre esses tipos de contagem está nas fórmulas aplicadas na contagem. São eles:

• *Projeto de desenvolvimento*: mede todas as funções que serão entregues com o projeto em sua primeira versão. É o tipo de contagem que será utilizada neste artigo;

• *Projeto de melhoria*: mede as funcionalidades alteradas, incluídas e excluídas ao projeto;

• *Aplicação*: mede as funções de uma aplicação já instalada.

2) Escopo da Contagem: é a fronteira da aplicação. Define as funções que serão incluídas em uma determinada contagem de pontos de função. No nosso estudo de caso, o escopo da contagem restringe-se às funcionalidades de pesquisa e cadastramento de cursos e alunos, conforme apresentado anteriormente;

3) Funções de Dado: consiste na contagem dos tipos de dados utilizados nas funcionalidades a serem desenvolvidas. Esses tipos de dados podem ser agrupados em um Arquivo Lógico Interno (ALI) que representa grupos de dados relacionados e reconhecidos pelo usuário, dentro da fronteira da aplicação; ou um Arquivo de Interface Externa (AIE) que representa dados referenciados pela aplicação mas mantidos dentro da fronteira de outra aplicação. No exemplo temos:

• **ALI:** Curso, Aluno

• **AIE:** Mensalidade

Para calcular a complexidade das funções de dados, deve-se contar o

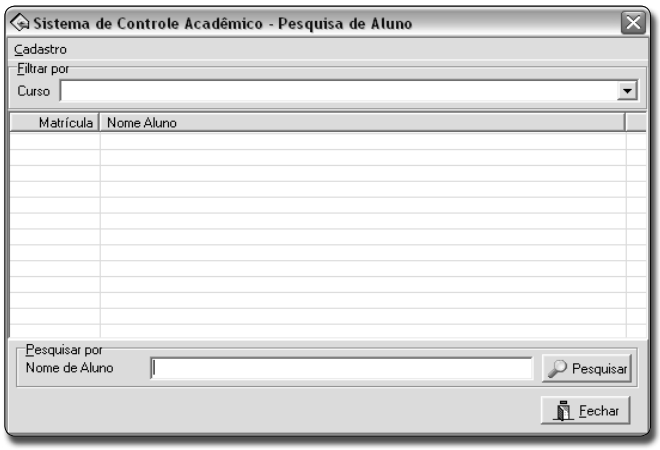


Figura 3. Janela de pesquisa de alunos

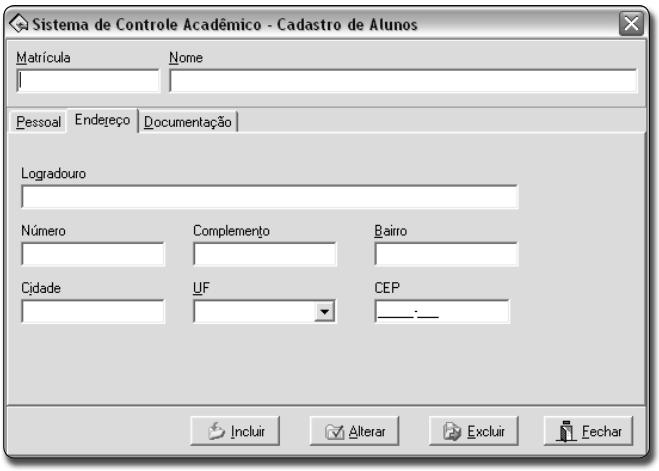


Figura 5. Janela de Cadastro de Alunos – Dados de Endereço

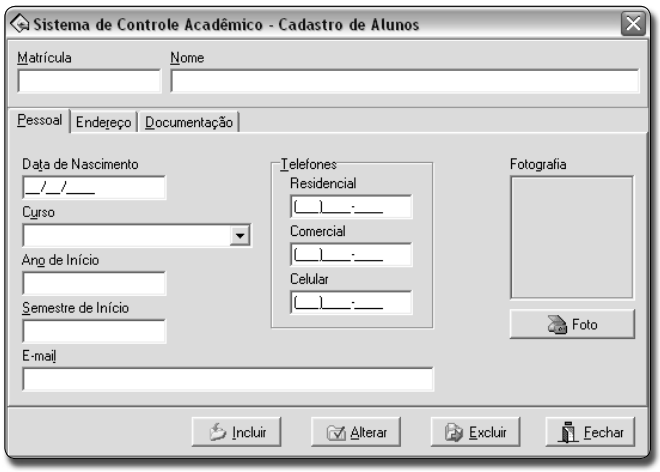


Figura 4. Janela de Cadastro de Alunos – Dados Pessoais

Figura 6. Janela de Cadastro de Alunos – Dados de Documentação

número de Tipos de Dados (TD) e Tipos de Registros (TR) de cada uma dessas funções. Um TD refere-se a um campo único reconhecido pelo usuário, sem repetição, ou seja, se um campo se repetir mais de uma vez numa interface, conta-se apenas uma vez.

Um TR refere-se a um subgrupo de tipos de dados, também reconhecidos pelo usuário e componente de um ALI ou AIE. No exemplo, os TDs estão apresentados nas **Tabelas 1** a **3**, representando os arquivos Curso, Aluno e Mensalidade, respectivamente. Como Curso apresenta dois tipos de registro (cursos de graduação e de pós-graduação com dados específicos), considera-se TR igual a 2.

Os demais possuem TR igual a 1. Se os cursos estivessem divididos em mais de um arquivo como, por exemplo, um arquivo para cursos de graduação e outros de pós, mesmo assim, consideraríamos para a contagem um único ALI com dois Tipos de Registros.

Identificados os TDs e TRs, pode-se calcular a complexidade de cada ALI/AIE através da **Tabela 4**. Analisando-a, observa-se que o ALI Curso é de complexidade baixa, pois possui dois TRs e cinco TDs, o ALI Aluno é de complexidade baixa, pois possui um TR e 23 TDs e, por fim, o AIE Mensalidade também é de complexidade baixa, pois possui um TR e dois TDs.

4) Funções de Transação: representam os processos elementares fornecidos pela aplicação ao usuário, onde um processo elemen-

TD	Tipo de Dado
1	Código do curso
2	Descrição do curso
3	Tipo do curso (1=Graduação / 2=Pós-Graduação)
4	Quantidade de períodos
5	Carga horária

Tabela 1. Tipos de Dados de Curso

TD	Tipo de Dado
1	Matrícula do aluno
2	Nome do aluno
3	Data de nascimento
4	Identificador do curso
5	Ano de início
6	Semestre de início
7	Email
8	Telefone residencial
9	Telefone comercial
10	Telefone celular
11	Foto
12	Logradouro
13	Número
14	Complemento
15	Bairro
16	Cidade
17	UF
18	CEP
19	CPF
20	Número identidade
21	Órgão expedidor
22	UF órgão expedidor
23	Data expedição

Tabela 2. Tipos de Dados de Aluno

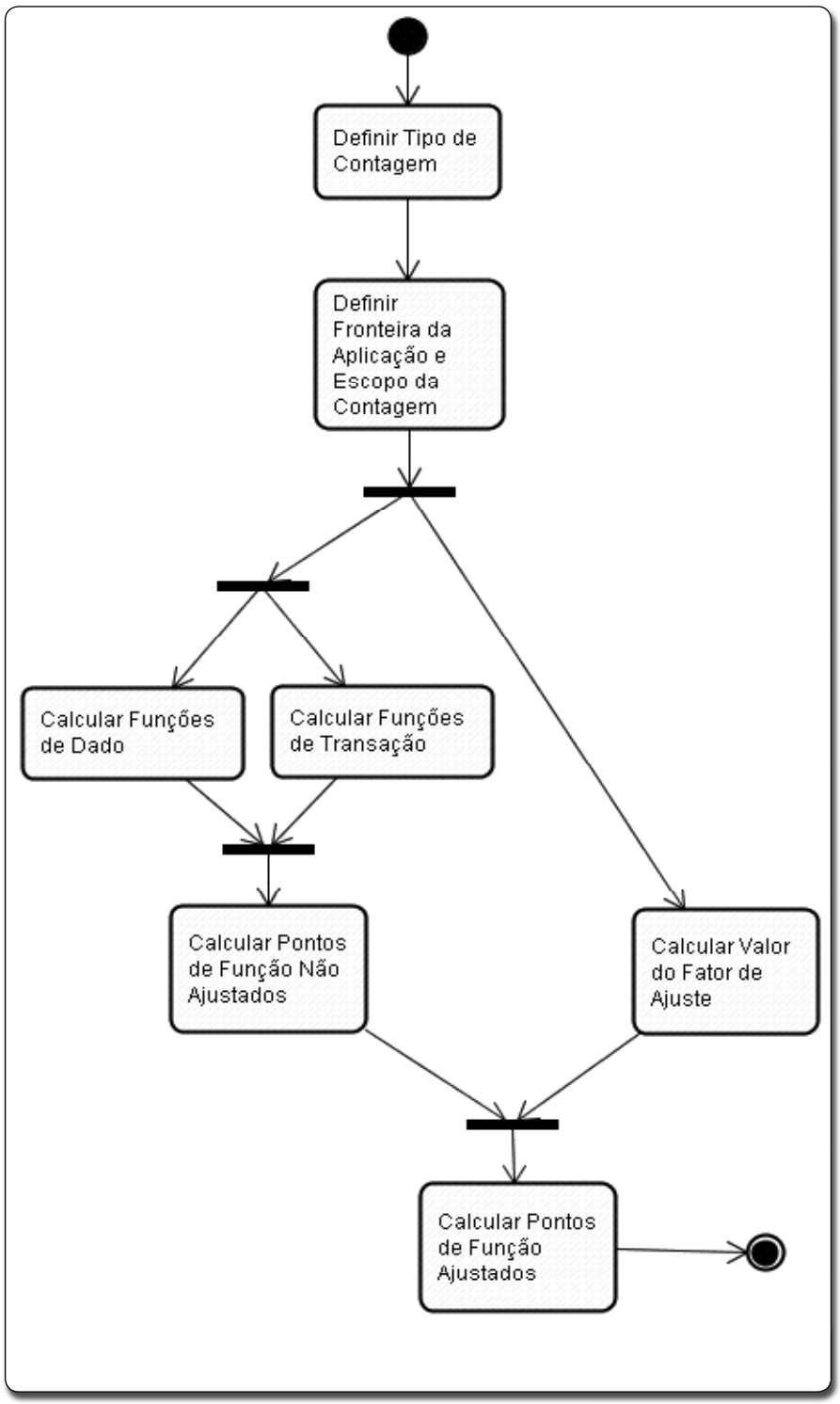


Figura 7. Processo de contagem de pontos de função.

tar identifica a menor unidade funcional do ponto de vista do usuário, e pode ser:

• *Entrada Externa (EE)*: para sua identificação devem ser analisados todos os processos elementares que processam dados vindos de fora da fronteira da aplicação e que atualizam um ou mais ALIs;

• *Saída Externa (SE)*: o processo gera dados para fora da fronteira da aplicação, tendo por objetivo principal apresentar dados ao usuário através de lógica de processamento que não apenas a recuperação de dados;

• *Consulta Externa (CE)*: processo que envia dados para fora da fronteira da aplicação, apresentando dados ao usuário por meio de uma simples recuperação de informações de ALI ou AIE.

No estudo de caso deste artigo, podem-se identificar os processos elementares conforme apresentado na **Tabela 5**.

As regras para definir a complexidade de EE, SE ou CE consideram o número de Arquivos Referenciados (AR) e o número de Tipo de Dados (TD) utilizados, sendo:

• AR: Para cada ALI lido ou mantido pela aplicação, ou AIE lido, conta-se um;

• TD:

- *Entrada Externa* (EE): contam-se os atributos que são atualizados e determina-se a complexidade conforme a **Tabela 6**;

- *Saída Externa* (SE): contam-se os atributos da saída, além de atributos calculados, e determina-se a complexidade conforme **Tabela 7**;

- *Consulta Externa* (CE): em um processo de entrada de dados, contam-se os atributos de seleção, além de mensagem ao usuário quando for o caso. Em um processo de saída de dados, contam-se os atributos de saída. Em processos de entrada e saída, somam-se os dois resultados. Determina-se a complexidade conforme a **Tabela 7**.

No estudo de caso deste artigo, consideram-se as análises apresentadas na **Tabela 8**.

Os processos elementares *Listar Cursos* e *Listar Alunos* consideraram TD igual a dois por ser esse o número de atributos exibidos como resultado na interface com o usuário. *Pesquisar Curso* e *Pesquisar Aluno* consideram AR igual a um, pois cada processo obtém informações de um único ALI e TD igual a três, pois, por ser um processo de entrada e saída, recebe o parâmetro a ser consultado e exibe os dados previstos nas respectivas telas de pesquisa (código e descrição para cursos e matrícula e nome para alunos).

Filtrar Alunos por Curso considera um AR a mais por ter que exibir os cursos cadastrados. O processo elementar *Exibir Mensagem de Curso Cadastrado* não considera nenhum AR, pois não obtém informações de nenhum ALI ou AIE e considera TD igual a um em função da mensagem exibida. Os processos elementares *Incluir, Alterar, Excluir* e *Consultar Alunos* consideram dois AR por atualizar o ALI Aluno e obter dados do ALI Curso.

O processo elementar *Exibir Cálculo de Valor da Mensalidade* considerou AR igual a um por ler o AIE Mensalidade e TD igual a dois, em virtude de ler o valor da mensalidade em função do tipo de curso.

5) Pontos de Função Não Ajustados: após identificar as funções de dados e os processos elementares, multiplica-se o total de ALI, AIE, EE, SE e CE pelo respectivo valor de ponto de função da tabela de complexidade (**Tabela 9**) para determinar o valor total que será o PF não ajustado.

A **Tabela 10** exibe o resultado final do cálculo de pontos de função não ajustados para o estudo de caso deste artigo.

6) Valor do Fator de Ajuste: representa a influência de requisitos técnicos e de qualidade no tamanho do software. É calculado com base em 14 características gerais de um sistema, onde cada uma delas deve ser analisada com relação ao seu nível de influência sobre o sistema e pontuada de 0 (nenhuma influência) a 5 (grande influência).

A descrição completa de cada característica e as tabelas para avaliação dos valores de pontuação, pode ser obtida no manual de Práticas de Contagem de Pontos de Função do IFPUG. Para este exemplo, as características foram descritas a seguir, bem como os valores considerados para cada uma delas:

1. Comunicação de Dados: considera se são utilizados recursos de comunicação

TD	Tipo de Dado
1	Tipo do curso (1=Graduação / 2=Pós-Graduação)
2	Valor Base da Mensalidade

Tabela 3. Tipos de Dados de Mensalidade

Tipos de Registro (TR)	Tipos de Dados (TD)		
	Abaixo de 20	20 a 50	Acima de 50
1	Baixa	Baixa	Média
2 a 5	Baixa	Média	Alta
Acima de 5	Média	Alta	Alta

Tabela 4. Complexidade das funções de dados (ALI e AIE)

Funcionalidade	Processo Elementar	Tipo
Pesquisa de Cursos	Listar Cursos	CE
	Pesquisar Curso	CE
Cadastramento de Cursos	Incluir Curso	EE
	Alterar Curso	EE
	Excluir Curso	EE
	Consultar Curso	CE
	Exibir Mensagem de Curso Cadastrado	CE
Pesquisa de Alunos	Listar Alunos	CE
	Pesquisar Aluno	CE
	Filtrar Alunos por Curso	CE
Cadastramento de Alunos	Incluir Aluno	EE
	Alterar Aluno	EE
	Excluir Aluno	EE
	Consultar Aluno	CE
	Exibir Cursos Cadastrados na Caixa de Combinação	CE
	Exibir Cálculo de Valor da Mensalidade	SE

Tabela 5. Processos Elementares identificados no estudo de caso

pela aplicação. Neste exemplo, será considerado o valor quatro, que é relativo a aplicações on-line, suportadas por algum protocolo de comunicação;

2. Processamento Distribuído: relativo à transferência de dados entre os componentes da aplicação. Foi considerado o valor um, uma vez que não realiza processamento distribuído, fazendo uso apenas de um sistema de gerenciamento de banco de dados;

3. Performance: determina como o tempo de resposta influencia a aplicação. Foi considerado zero, uma vez que não foi indicada nenhuma restrição a esse respeito;

4. Configuração Altamente Utilizada: define o nível de restrições impostas pelo usuário. Também foi considerado o valor zero uma vez que nenhuma restrição foi imposta;

5. Volume de Transações: refere-se ao volume de informações processados pela aplicação. Considerou-se o valor um, uma vez que é previsto um número de transações mais elevado em alguns períodos específicos, como a cada período de matrículas de alunos;

6. Entrada de Dados On-line: considera a quantidade de transações feitas on-line. Foi considerado o valor cinco uma vez que o sistema apresenta todas as transações como sendo desse tipo;

7. Eficiência do Usuário Final: refere-se a facilidades oferecidas ao usuário final, como ajuda on-line, auxílio à navegação por teclas de função, menus, dentre outros. Foi considerado o valor um por apresentar algumas poucas características desse tipo, como a exibição dos cursos em caixa de combinação para facilitar a seleção do usuário no cadastramento de alunos;

8. Atualização On-line: define se os arquivos lógicos internos são atualizados on-line. Foi utilizado o valor três, pois todos os arquivos internos são atualizados dessa forma;

9. Complexidade de Processamento: determina a complexidade de processamento

Função	Baixa	Média	Alta
Consulta Externa (CE)	3	4	6
Entrada Externa (EE)	3	4	6
Saída Externa (SE)	4	5	7
Arquivo de Interface Externa (AIE)	5	7	10
Arquivo Lógico Interno (ALI)	7	10	15

Tabela 9. Contribuição de ponto de função por complexidade

Arquivos Referenciados (AR)	Tipos de Dados (TD)		
	Abaixo de 5	5 a 15	Acima de 15
0 ou 1	Baixa	Baixa	Média
2	Baixa	Média	Alta
Acima de 2	Média	Alta	Alta

Tabela 6. Complexidade de Entrada Externa (EE)

Arquivos Referenciados (AR)	Tipos de Dados (TD)		
	Abaixo de 6	6 a 19	Acima de 19
0 ou 1	Baixa	Baixa	Média
2 ou 3	Baixa	Média	Alta
Acima de 3	Média	Alta	Alta

Tabela 7. Complexidade de Saída Externa (SE) e Consulta Externa (CE)

Processo Elementar	Tipo	AR	TD	Complexidade
Listar Cursos	CE	1	2	Baixa
Pesquisar Curso	CE	1	3	Baixa
Incluir Curso	EE	1	5	Baixa
Alterar Curso	EE	1	5	Baixa
Excluir Curso	EE	1	5	Baixa
Consultar Curso	CE	1	5	Baixa
Exibir Mensagem de Curso Cadastrado	CE	0	1	Baixa
Listar Alunos	CE	1	2	Baixa
Pesquisar Aluno	CE	1	3	Baixa
Filtrar Alunos por Curso	CE	2	3	Baixa
Incluir Aluno	EE	2	23	Alta
Alterar Aluno	EE	2	23	Alta
Excluir Aluno	EE	2	23	Alta
Consultar Aluno	CE	2	23	Alta
Exibir Cursos Cadastrados na Caixa de Combinação	CE	1	1	Baixa
Exibir Cálculo de Valor da Mensalidade	SE	1	2	Baixa

Tabela 8. Complexidade dos processos elementares identificados no estudo de caso.

Processo Elementar	Tipo	Complexidade	Pontos Função
Curso	ALI	Baixa	7
Aluno	ALI	Baixa	7
Mensalidade	AIE	Baixa	5
Listar Cursos	CE	Baixa	3
Pesquisar Curso	CE	Baixa	3
Incluir Curso	EE	Baixa	3
Alterar Curso	EE	Baixa	3
Excluir Curso	EE	Baixa	3
Consultar Curso	CE	Baixa	3
Exibir Mensagem de Curso Cadastrado	CE	Baixa	3
Listar Alunos	CE	Baixa	3
Pesquisar Aluno	CE	Baixa	3
Filtrar Alunos por Curso	CE	Baixa	3
Incluir Aluno	EE	Alta	6
Alterar Aluno	EE	Alta	6
Excluir Aluno	EE	Alta	6
Consultar Aluno	CE	Alta	6
Exibir Cursos Cadastrados na Caixa de Combinação	CE	Baixa	3
Exibir Cálculo de Valor da Mensalidade	SE	Baixa	4
Total de Pontos de Função			80

Tabela 10. Identificação dos processos elementares e suas complexidade

das funcionalidades da aplicação. Foi considerado o valor zero, por ser uma aplicação bastante simples;

10. Reutilização: identifica se o código foi projetado para ser reaproveitado em outras aplicações. O valor zero foi considerado por essa não ter sido uma preocupação no projeto;

11. Facilidade de Instalação: considera se existem ferramentas de conversão e instalação da aplicação. Também foi considerado o valor zero por essa preocupação não ter sido levada em consideração no estudo de caso;

12. Facilidade de Operação: considera aspectos de segurança e recuperação de informações. Como isso também não foi considerado, é assumido o valor zero;

13. Múltiplos Locais: se a aplicação foi projetada e desenvolvida para ser utilizada em diferentes locais. Essa restrição também não foi considerada e assumido o valor zero;

14. Facilidade de Mudanças: se a aplicação foi projetada para facilitar mudanças na lógica de processamento ou em suas estruturas de dados. O valor zero foi utilizado nesse caso.

Somam-se então essas pontuações para obter o nível total de influência (TDI – *Total Degree of Influence*). Daí basta aplicar a seguinte fórmula para obter o valor do fator de ajuste (VAF – *Value Adjustment Factor*):

```
Valor do Fator de Ajuste (VAF) =
  (TDI x 0,01) + 0,65
```

No caso do exemplo deste artigo:

```
VAF = (15 x 0,01) + 0,65 = 0,8
```

Atualmente esse é um passo opcional do processo de contagem. Muitas organizações desconsideram o fator de ajuste e usam apenas a medição dos pontos de função não ajustados. Isso se deve ao fato de que as características gerais de um sistema, apresentadas anteriormente, são consideradas desatualizadas ou incompletas em relação às tecnologias atualmente utilizadas, fazendo com que sua utilidade seja questionada.

Isso fica mais evidenciado quando a norma ISO/IEC 14143 considera apenas os pontos de função não ajustados em suas estimativas de medição funcional.

7) Pontos de Função Ajustados: Os PFs ajustados são calculados a partir dos PFs não ajustados multiplicado pelo fator de ajuste. No caso de processos de desenvolvimento, funções para conversão de dados também devem ser consideradas e sua complexidade em pontos de função deve ser adicionada aos pontos de função não ajustados.

Assim, a seguinte fórmula para processos de desenvolvimento é aplicada:

```
PF Desenvolvimento = (PF Não Ajustado +
 PF Conversão de Dados) * Fator Ajuste (VAF)
```

Para o caso do exemplo utilizado, como não foi considerada a conversão de dados:

```
PF Desenvolvimento = (80 + 0) * 0,8 = 64 PFs
```

Estimativas de esforço e prazo a partir da Contagem de Pontos de Função

Pontos de Função têm por objetivo a medição do tamanho funcional de uma aplicação. Entretanto, não é incomum que desenvolvedores queiram estimar esforço e prazos a partir do número de pontos de função. Deve-se tomar cuidado com essa situação, pois a produtividade de diferentes equipes de desenvolvimento pode variar consideravelmente, inclusive em função das tecnologias utilizadas.

A falta de conhecimento da produtividade da equipe pode fazer com que as estimativas de esforço sejam erradas, sem que pontos de função sejam os culpados por isso. Outra situação preocupante é utilizar tabelas de produtividade por ponto de função disponíveis na internet, onde os dados ali apresentados podem não refletir a capacidade de produção de qualquer equipe de desenvolvimento e, muitas vezes, não levam em consideração as características de diferentes tecnologias que podem ser utilizadas.

Portanto, o conhecimento da produtividade de cada equipe é fundamental para estimativas de esforço e prazo a partir da contagem de pontos de função. No estudo de caso deste artigo, vamos considerar hipoteticamente a situação de que um ponto de função é desenvolvido em uma hora de trabalho de um desenvolvedor (1 homem/hora).

Assim, o esforço para esse projeto, considerando-se os pontos de função ajustados, seria de 64 horas de trabalho de um único desenvolvedor. Outra situação que merece atenção é utilizar o valor real de produtividade de um desenvolvedor, uma vez que dificilmente alguém tem produtividade igual a sua carga horária total de trabalho.

Neste exemplo hipotético, se considerarmos que um desenvolvedor produz cerca de 35 horas reais por semana, o estudo de caso deste artigo levaria menos de duas semanas para ser desenvolvido, se conduzido por um único desenvolvedor.

Conclusão

Contagem de Pontos de Função representa uma técnica de cálculo do tamanho de uma aplicação, não de esforço para seu desenvolvimento. Sua correta utilização, baseada nas regras descritas no manual de Práticas de Contagem de Pontos de Função do IFPUG, pode ajudar na medição da funcionalidade de sistemas e apoiar estimativas de esforço e prazo baseadas na produtividade conhecida de uma equipe de desenvolvimento.

Pode-se ainda encontrar diversas ferramentas ou planilhas eletrônicas de diferentes fornecedores para apoiar os procedimentos de cálculo de pontos de função de forma a facilitar o processo de contagem. Este artigo procurou apresentar de forma prática as principais regras de contagem de pontos de função através de um estudo de caso prático. ●

Referências

Site do IFPUG: International Function Point Users Group
www.ifpug.org

Site do BFPUG-Brazilian Function Point Users Group
www.bfpug.com.br

Livro Análise de Pontos de Função – Medição, Estimativas e Gerenciamento de Projetos de Software
3ª. edição, *Carlos Eduardo Vazquez, Guilherme Siqueira Simões e Renato Machado Albert.*
Editora Érica, 2003.

Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback

POR QUE A ALOG
É LÍDER EM HOSTING GERENCIADO?

- Dois Data Centers próprios – RJ e SP
- Projetos de Hosting customizados
- Foco em serviços de Data Center
- Gerência total dos servidores
- 850 clientes corporativos
- 8 mil m² de área construída
- Destaque do Ano no segmento Internet Serviços no Anuário Telecom 2007

EMPRESA CERTIFICADA ISO 9001

EQUIPE CERTIFICADA EM:

PMI	PMP (PROJECT MANAGEMENT PROFESSIONAL)
ITIL	ITIL FOUNDATION
MICROSOFT	MCP (MICROSOFT CERTIFIED PROFESSIONAL), MCDST (MICROSOFT CERTIFIED DESKTOP SUPPORT TECHNICIAN), MCSE (MICROSOFT CERTIFIED SYSTEM ENGINEER)
CISCO	CCNA (CISCO CERTIFIED NETWORK ASSOCIATE) CCDA (CISCO CERTIFIED DESIGN ASSOCIATE)
RED HAT	RHCE (RED HAT CERTIFIED ENGINEER)
LPI INSTITUTE	LPI-1 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 1) LPI-2 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 2)
GIAC	GSEC (GIAC SECURITY ESSENTIALS GOLD)
CHECKPOINT	CCSA (CHECKPOINT CERTIFIED SECURITY ADMINISTRATOR)

ALOG
DATA CENTERS DO BRASIL

www.alog.com.br | 0800 282 3330

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Usando todo o poder do TDataSetProvider

Usufrua de todos os recursos do DataSetProvider em suas aplicações

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial da revista ClubeDelphi e WebMobile. Mantém o blog Delphi to Delphi (www.delphi-todelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Sem dúvida nenhuma, um dos componentes mais poderosos da VCL é o *DataSetProvider*, que, além de prover dados a toda a aplicação, é capaz de auxiliar em uma série de funcionalidades no sistema.

Veremos o que há de mais interessante nesse componente e dicas avançadas para usufruir ao máximo de suas propriedades e métodos. Em conjunto com os componentes *DBExpress* o *DataSetProvider* é capaz de realizar inúmeras tarefas. Veremos nesse artigo:

- Introdução ao DataSetProvider;
- Como usar o UpdateMode e ProviderFlags;
- Configurando dinamicamente o UpdateMode e ProviderFlags;
- Uso de Constraints;

Criaremos diversos exemplos para entendermos corretamente cada funcionalidade do *DataSetProvider*;

Entendendo o DataSetProvider

Basicamente o *DataSetProvider* é responsável por enviar e receber os *Data Packets* da aplicação cliente para a o servidor de dados. Os *Data Packets* são os pacotes de dados trafegados na rede em uma aplicação duas camadas ("two-tier" ou "client/server") ou "n" camadas ("n-tier"). Toda e qualquer requisição feita pela aplicação cliente é enviada ao *DataSetProvider* que por sua vez se encarrega de solicitar os dados ao servidor de aplicação. O *result set*, ou seja, o resultado da requisição é empacotado ("Data Packets") e enviado de volta à aplicação cliente. Qualquer exceção levantada, seja na requisição ou no recebimento dos dados, é retornada a aplicação cliente.

Pode-se dizer que a utilização mais comum do *DataSetProvider* é feita em conjunto dos componentes da paleta *DBExpress* acrescido do componente *ClientDataSet*. Um exemplo de uso pode ser visto na **Figura 1**.

O que pouca gente sabe é que o *DataSetProvider* pode se tornar mais do que um simples componente de conexão com o banco de dados por ser farto de propriedades e eventos.

Constraints e DataSetProvider

Como sabemos, as *constraints* de uma aplicação podem ser inseridas de diversas formas em uma aplicação. Em aplicações *two-tier*, por exemplo, temos apenas dois lugares onde elas podem ser incluídas: no lado *servidor* ou na *aplicação cliente*. No servidor devemos inseri-las diretamente no SGBD através de *Stored Procedures, domains, rules, triggers etc*. É uma excelente idéia, pois centralizamos nossas regras de negócios em um único lugar. Porém, corremos o risco de ficarmos presos ao banco de dados que estamos trabalhando por conta da linguagem empregada no SGBD. Em outras palavras quando programamos diretamente no banco necessitamos usar a linguagem de programação SQL, também chamada ANSI, para criar nossas próprias instruções. Isso pode se tornar uma dor de cabeça em uma eventual mudança de banco de dados ou mesmo se o produto vier a ser comercializado com BD's diferentes. Imagine um software que precisa ser capaz de se conectar aos bancos Interbase/Firebird, SQL Server 2005 Express e Oracle. De início já teremos algumas complicações entre Interbase e Firebird dependendo da versão de ambos, haja vista que muitas modificações no BD foram feitas ao longo dos anos. Por isso, da mesma forma que o esquema é interessante, também pode tornar-se um trauma.

Por outro lado é possível incluir *constraints* diretamente na aplicação cliente, porém temos uma enorme desvantagem: a *decentralização* de nossas regras de negócios. Elas precisam ser refeitas para cada nova aplicação e espalhadas por todo o código fonte ou em *Units*, componentes ou *dll's*. Essa prática é altamente perigosa, pois podem ocorrer momentos em que uma regra de negócio pode não ser alterada por esquecimento dos membros da equipe de desenvolvimento, o que acarretaria em inconsistência dos dados.

Em aplicações *n-tier* (multi-camadas) podemos recorrer ao poderoso DataSnap e programar nossas regras de negócio diretamente no servidor de aplicação, tornando a aplicação mais consistente e inteligente. Essas regras residem no código fonte do servidor de aplicações, e ficam centralizadas.

Resumindo, uma das melhores alternativas certamente é fazer uso desta última opção, ou seja, utilizar um servidor de aplicação que fará o intercâmbio entre *bando de dados* (SGBD) e *aplicação cliente*, pois além de propiciar maior controle sobre as regras de negócios, ainda podemos disponibilizar a aplicação para acesso remoto através da internet.

Propriedades do TField

Algumas propriedades to *TField* são automaticamente passadas da aplicação servidora para o cliente e a maioria dos valores são determinados em tempo de projeto baseando-se na estrutura da tabela no SGBD. Podemos usufruir de alguns recursos somente disponíveis nas propriedades do *TField*, seja em aplicações *single tier*, *two-tier* ou *n-tier* e que não estão disponíveis no DataSnap. Em suma, é possível criarmos pequenas *constraints* diretamente na aplicação cliente e que podem ser facilmente configuradas. Isso é bastante interessante, visto que reduz drasticamente a quantidade de desnecessária de código e ainda permite que sejam personalizadas as mensagens de exceção geradas. Essas propriedades são:

- **Read Only**: Como o próprio nome indica, podemos "setar" um campo como somente leitura;
- **Required**: Essa propriedade estando configurada como True, faz com que a aplicação obrigue o usuário final a digitar um valor nela. Grosseiramente falando, campos Not Null no banco de dados são fortes candidatos a serem campos Required, tanto é, que quando adicionamos um campo Not Null aos Field's Editor's do DataSet, este já vem como True em sua propriedade;
- **DefaultExpression**: Valor padrão para inserção no banco de dados. Aqui podemos configurar um valor que será gravado no banco caso esteja vazio;
- **CustomConstraint**: É exatamente nessa propriedade que definimos nossa constraint. Digitamos a expressão e posteriormente o sistema fará a validação. Ex.: X > 100 and X < 200;

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli que mostra como trabalhar com constraints e ClientDataSet.

www.devmedia.com.br/articles/viewcomp.asp?comp=567&hl=

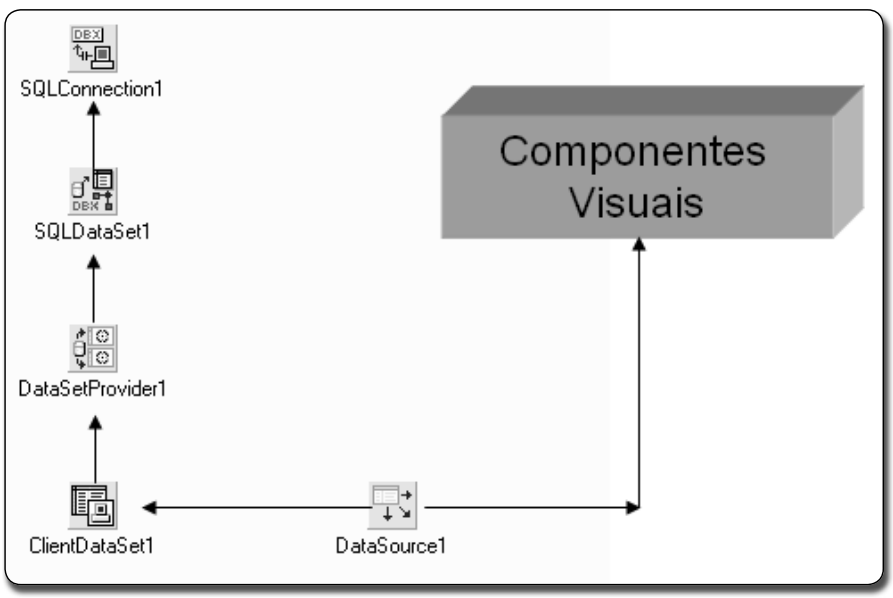


Figura 1. Exemplo de uso do componente DataSetProvider

• ConstraintErrorMessage: Por último a mensagem de erro que será exibida ao usuário final da aplicação;

Devemos nos lembrar que isso tudo só é possível com campos persistentes, ou seja, com tipificação de campos no projeto. Esse recurso é utilizado quando adicionamos os campos da tabela no *Field's Editor* do *TDataSet*, do contrário não será possível.

Outras duas informações são importantíssimas. A primeira é que as mensagens de erro são resultantes da violação das regras impostas nos *TField's* do *TDataSet*, e conseqüentemente gerarão exceções. Se outras regras também forem definidas no servidor de dados, ou seja, diretamente no SGBD, estas também levantarão exceções, porém somente uma das duas mensagens será exibida. A segunda informação diz respeito à alteração de regras no SGBD. Algumas das regras de negócios impostas diretamente no BD precisam ser replicadas para a aplicação, como é o caso de campos requeridos ("Require's"). Caso modifique de *True* para *False* ou vice-versa em um determinado campo, este precisa ser removido ou alterado em tempo de projeto. Vejamos um exemplo:

Abra o Delphi e crie uma nova aplicação usando *File|New>Application* e salvando-a como "Properties.dpr" e o formulário principal como "Principal.pas". Insira no formulário um componente *SQLConnection* e um *TSQLDataSet*, ambos da paleta *dbExpress*. Dê um clique duplo no *SQLConnection* e crie uma nova conexão com o banco de dados *Employee.fdb* presente no diretório de instalação do Firebird, normalmente em *C:\Arquivos de Programas\Firebird\Firebird_Versão\examples\empbuild\Employee.fdb*. Conecte o *SQLDataSet* ao *SQLConnection* usando a propriedade *SQLConnection* e em seguida digite a instrução SQL a seguir para selecionar os dados da tabela *SALES* do banco.

```
SELECT * FROM SALES
```

Arraste agora um *DataSetProvider* e um *ClientDataSet* da paleta *Data Access*. Ligue o *DataSetProvider* ao *SQLDataSet* pela propriedade *DataSet* e o *ClientDataSet* ao *Provider* usando *ProviderName*. Abra seu gerenciador de banco de dados preferido e visualize a estrutura da tabela *SALES* (**Figura 2**).

Note que os campos *PO_NUMBER, CUST_NO, ORDER_STATUS, ORDER_DATE, QTY_ORDERED, TOTAL_VALUE, DISCOUNT* e *ITEM_TYPE*, estão marcados como *Not Null*, ou seja, não permitem que sejam gravados sem nenhum valor. Experimente adicionar todos os campos da tabela *SALES* ao *Field's Editor* do *ClienteDataSet* usando o menu de contexto e a opção *Add all fields*. Em seguida clique em cada campo e observe a propriedade *Required*. Perceba que cada campo *Not Null* teve sua propriedade *Required* modificada para *True*, isso significa que o sistema obrigará o usuário a inserir um valor nesses campos (**Figura 3**).

Fields | Constraints | Indices | Dependencies | Triggers | Data | Master/Detail View

AGED COMPUTED BY ((ship_date - order_date))

#	PK	FK	UNQ	Field Name	Field Type	Size	Not Null
1	PK1			PO_NUMBER	CHAR	8	☒
2		FK		CUST_NO	INTEGER		☒
3		FK		SALES_REP	SMALLINT		☐
4				ORDER_STATUS	VARCHAR	7	☒
5				ORDER_DATE	TIMESTAMP		☒
6				SHIP_DATE	TIMESTAMP		☐
7				DATE_NEEDED	TIMESTAMP		☐
8				PAID	CHAR	1	☐
9				QTY_ORDERED	INTEGER		☒
10				TOTAL_VALUE	DECIMAL	9	☒
11				DISCOUNT	FLOAT		☒
12				ITEM_TYPE	VARCHAR	12	☒
13				AGED	NUMERIC	18	☐

Figura 2. Estrutura da tabela SALES do banco Employee.fdb

Propriedade adicionais do TField usadas pelo DataSetProvider

Agora vejamos algumas das propriedades mais importantes em um *TField* e que são utilizadas fortemente pelo *DataSetProvider*. A propriedade *Options* do *DSP* ("DataSetProvider") suporta a passagem de propriedades adicionais do *TField's*. Quando configuramos como *True* a propriedade *poIncFieldProps*, podemos passar propriedades do *TField* para o *DSP*. Essas propriedades são:

- *Alignment;*
- *Currency;*
- *DisplayFormat;*
- *DisplayLabel;*
- *DisplayWidth;*
- *EditFormat;*
- *EditMask;*
- *MaxValue;*
- *MinValue;*
- *Visible;*

O que poucos sabem, é que podemos configurar outras propriedades do *DSP* capazes de nos auxiliar em diversas tarefas. Ainda na propriedade *Options* vejamos algumas delas:

- *poReadOnly:* Desabilita qualquer alteração no DataSet ligado a ela;
- *poDisableInserts:* Desabilita a inserção de dados na tabela;
- *poDisableEdits:* Dasativa a edição dos registros;
- *poDisableDeletes:* Não permite a exclusão de registros;

Retorne ao projeto criado anteriormente e desenhe uma tela semelhante à **Figura 4**. Selecione o *DataSerProvider* e configure a propriedade *Options>poReadOnly* para *True*, ative o *ClientDataSet* e execute a aplicação. Perceba que o *DBNavigator* permite apenas a navegação dos registros (**Figura 5**) e não mais as demais opções.

Seguindo a mesma linha de raciocínio, agora modifique a propriedade *Options>DisableEdits*, salve o projeto e execute. Na seqüência, experimente clicar no botão de alteração e note o erro (**Figura 6**).

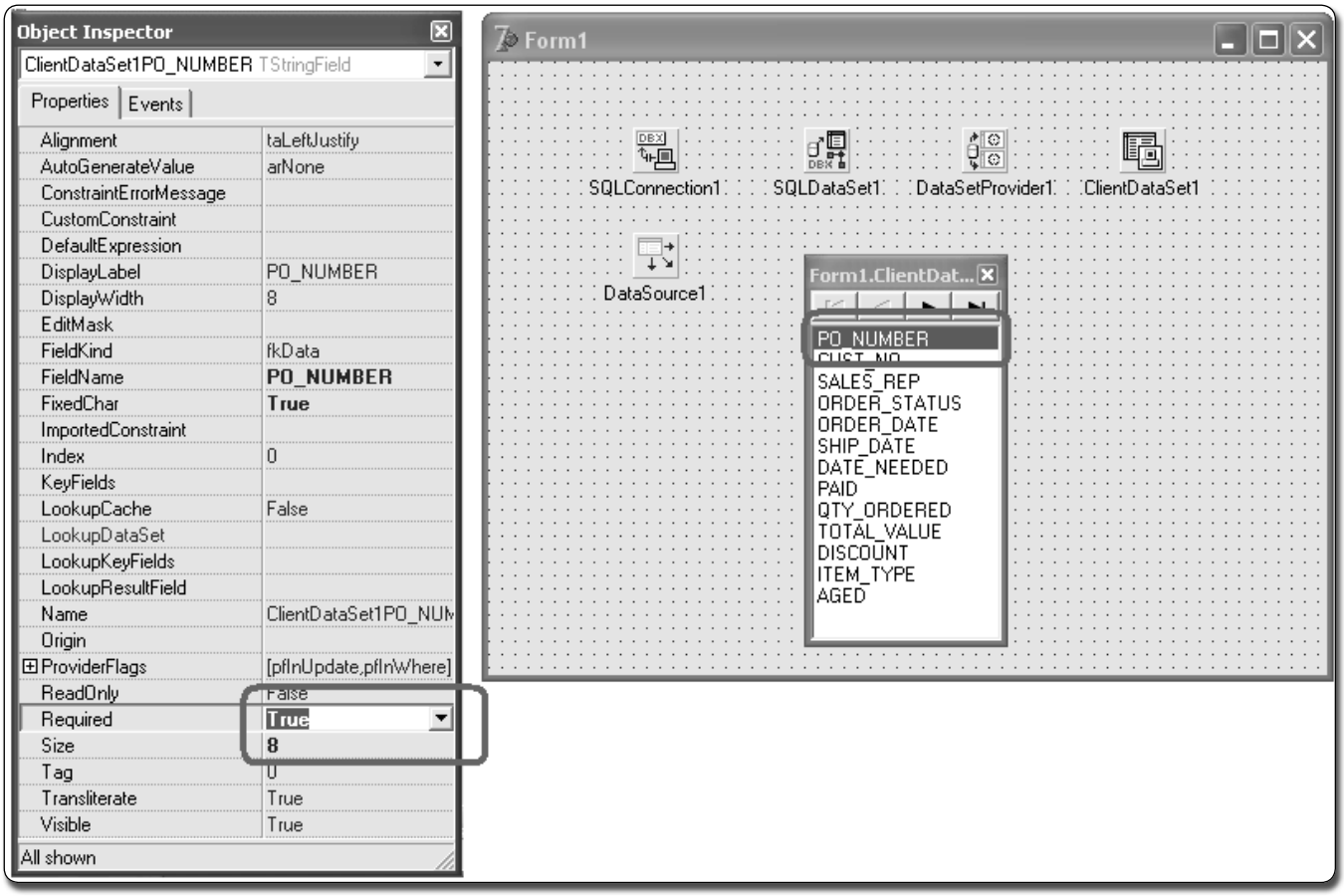


Figura 3. Configuração do TField adicionado automaticamente

Isso acontece, porque informamos ao *DSP* que não é permitida a alteração de registros por parte da aplicação cliente, logo a mensagem "ClientDataSet1: Modifications are note allowed.", traduzindo, "ClientDataSet1: Modificações não são permitidas.".

Usando os eventos do DSP

Podemos utilizar três eventos para validar *constraints* em uma aplicação usando o *DataSetProvider*. São eles:

• *BeforeUpdateRecord:* Acontece antes que os dados sejam atualizados na aplicação remota;

• *OnUpdateData:* Acontece ao aplicar os dados recebidos pela aplicação cliente, no servidor de dados;

• *OnUpdateError:* Quando ocorrem erros ao tentar efetuar o *Update*.

Usamos o evento *BeforeUpdateRecord* quando necessitamos fazer a validação individual dos registros enviados ao servidor. Também podemos modificar dados recebidos da aplicação cliente. A assinatura completa do evento podemos ver a seguir, juntamente com a descrição de cada parâmetro:

```
BeforeUpdateRecord(Sender: TObject;
  SourceDS: TDataSet; DeltaDS:
    TClientDataSet;
  UpdateKind: TUpdateKind;
  var Applied: Boolean);
```

• *Sender: DataSetProvider* que disparou o evento;

• *SourceDS:* Grosseiramente falando é a fonte de dados, os dados em si;

• *DeltaDS:* Pacote de dados enviado pela aplicação cliente;

• *UpdateKind:* Tipo de update, atualização, que será feita. Os valores possíveis são: *ukInsert, ukModify e ukDelete;*

• *Applied:* Indica se as modificações serão aplicadas.

Usando o evento BeforeUpdateRecord

Vejamos um exemplo prático. No evento *BeforeUpdateRecord* digite o código da **Listagem 1**. Nesse caso estamos verificando primeiramente se o tipo de atualização não é uma instrução para *deletar* o registro, pois caso seja não há necessidade em se validar um registro que será apagado. Em seguida testamos se o valor recebido é diferente de *Null*("VarIsNull") e diferente de *vazio*("VarIsEmpty"). Por fim, verificamos se o valor do campo *ORDER_DATE* não é superior ao *SHIP_DATE*. Havendo quaisquer divergência uma exceção é levantada.

Usando o evento OnUpdateData

O evento *OnUpdateData* ocorre uma vez ao iniciar a aplicação dos dados no servidor, ou seja, no recebimento dos registros Delta do *ClientDataSet*. É nesse momento que interceptamos as modificações e aceitamos ou até modificamos os dados se necessário. Aqui também podemos enviar mensagens à aplicação cliente caso seja necessário.

Veja a assinatura do evento logo em seguida:

```
OnUpdateData(Sender: TObject; DataSet:
  TClientDataSet);
```

- *Sender: DSP* que disparou o evento;
- *DataSet:* Pacote de dados, *Data Packet;*

No caso do evento anterior, *BeforeUpdateData*, não temos acesso total aos dados que estão vindo, e sim a um registro em particular. Já no caso do *OnUpdateData*, recebemos o *DataSet*, que contém todos os dados alterados e não alterados. Sendo assim, podemos ler individualmente o status de cada registro da tabela através do *Delta*. Insira mais um componente *ClientDataSet*("Delta") e um *DataSource*("DataSource1") ao sistema. Desconecte o *ClientDataSet* do *provider* limpando a propriedade *ProviderName*. Conecte o *DataSource1* ao *Delta*. Insira um *Button* e um *DBGrid*. Ligue o *DBGrid* ao *dsDelta* e digite o código da **Listagem 2** no evento *OnClick* do *Button* (**Figura 7**).

O que estamos fazendo é muito simples. Apenas atribuímos à propriedade *Data* do *Delta* o conteúdo da propriedade *Delta* do *ClientDataSet1*, que contém os dados originais acrescido das alterações efetuadas. Execute o programa, efetue algumas alterações na tabela e em seguida clique no botão *Delta*. Perceba que o 2º *DBGrid* mostra apenas alguns registros. Eles fazem parte do *Data Packet*, ou seja, o pacote de dados que será enviado ao *DSP* para montagem das instruções de inclusão, alteração e exclusão dos registros na base de dados. São assim que chegam os dados ao *DSP* (**Figura 8**).

O primeiro registro com o código *V91E0210*, foi marcado para exclusão. Já

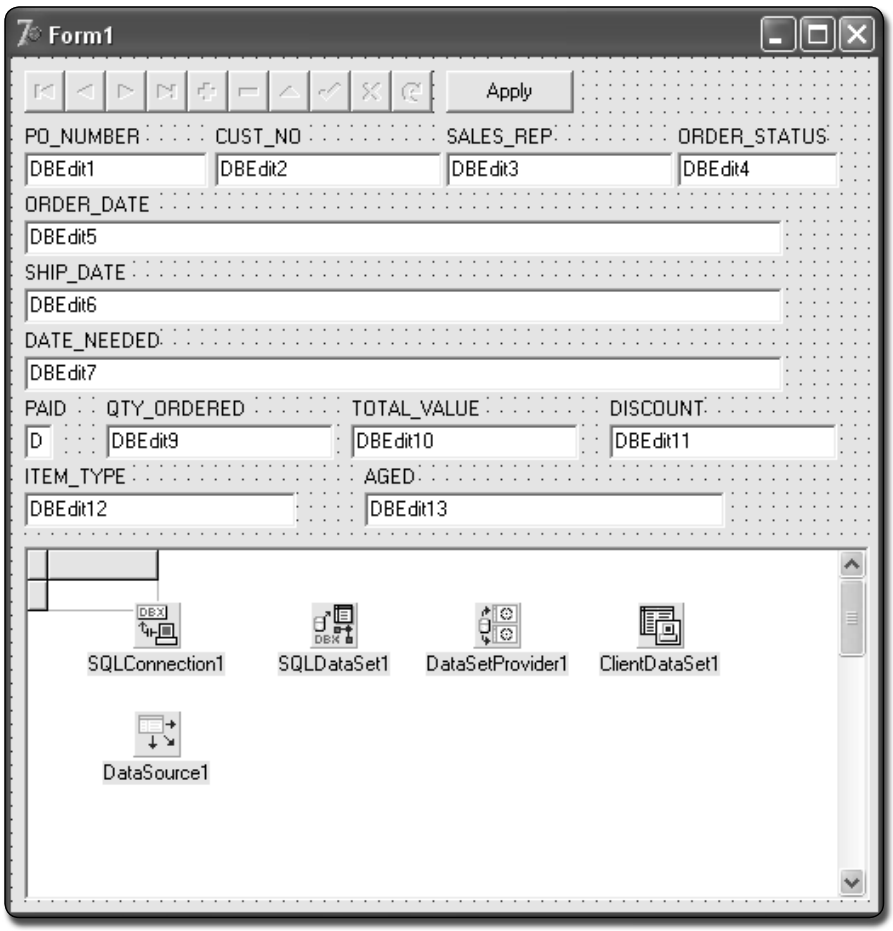


Figura 4. Exemplo de tela

Listagem 1. Código do evento BeforeUpdateRecord

```
procedure TForm1.DataSetProvider1BeforeUpdateRecord(
  Sender: TObject; SourceDS: TDataSet; DeltaDS:
  TClientDataSet; UpdateKind: TUpdateKind;
  var Applied: Boolean);
begin
  if UpdateKind <> ukDelete then
   if ((VarIsNull(DeltaDS.FieldByName('ORDER_DATE')
     .NewValue)  = False) and
     (VarIsEmpty(DeltaDS.FieldByName('ORDER_DATE')
     .NewValue) = False)) then
     if DeltaDS.FieldByName('ORDER_DATE').NewValue >
       DeltaDS.FieldByName('SHIP_DATE').OldValue then
         raise Exception.Create('Data do pedido não pode ser
           superior a data de compra.');
end;
```

Figura 5. DBNavigator apenas com funções de navegação

Figura 6. Exceção gerada pelo DSP

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli que mostra como funciona a arquitetura do Data e Delta do ClientDataSet.

www.devmedia.com.br/articles/viewcomp.asp?comp=5717&hl=

Nota do DevMan

Se você quiser mostrar mensagens personalizadas de erros em seu DSP, basta fazer uso dos eventos OnEditError, OnPostError e OnDeleteError presentes no ClientDataSet. Para isso basta acessar o evento que deseja modificar e inserir a mensagem desejada. Ex:

```
procedure TForm1.ClientDataSet1EditError
(DataSet: TDataSet;
  E: EDatabaseError; var Action:
TDataAction);
begin
  MessageDlg('Não são permitidas
   alterações nessa tabela.',
   mtInformation, [mbOk], 0);
  Action := daAbort;
end;
```

Note que estamos alterando o parâmetro Action para daAbort, dessa forma a mensagem original do ClientDataSet não será exibida, mostrando apenas nossa caixa de diálogo.

o segundo registro, *V92J1003*, teve o campo *CUST_NO* alterado de *1010* para *300*, por isso aparece na visualização apenas o campo *CUST_NO* alterado. Por fim os dois últimos registros foram inclusos na tabela. A partir do momento que o método *ApplyUpdates* do *ClientDataSet* for chamado, o *DSP* fará a montagem das instruções SQL e as enviará ao servidor de dados, que por sua vez executará todo o processo de atualização.

Usando o evento OnUpdateError

Nas seções anteriores vimos que é possível incluir mensagens personalizadas na aplicação para lidar com erros do sistema. Aqui veremos como trabalhar com o evento *OnUpdateError*. Nele podemos interceptar o erro, processá-lo, corrigi-lo e até mesmo modificá-lo para que o usuário final tenha mais confiança no aplicativo e entenda melhor tudo que está acontecendo. A seguir encontramos uma breve descrição de cada parâmetro do evento, bem como sua assinatura:

```
OnUpdateError(Sender: TObject;
  DataSet: TClientDataSet;
  E: EUpdateError;
  UpdateKind: TUpdateKind;
  var Response: TResolverResponse);
```

- *Sender: DataSerProvider* que disparou o erro;
- *DataSet: DataSet* temporário para acessar o erro;
- *E:* Objeto de exceção;
- *UpdateKind:* Tipo de *update;*
- *Response:* Ação de resposta ao erro;

Assim como os eventos que já vimos, aqui nós podemos usar os valores *NewValue*("Novo Valor") e *OldValue*("Valor Anterior") do *TField*. Também é possível utilizar o valor *CurValue* ("Valor atual"), que indica o valor atual do banco de dados. Com isso podemos visualizar o valor original, valor armazenado atualmente e o valor que deverá ser aplicado.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli que mostra como a arquitetura os Data Packets no ClientDataSet.

www.devmedia.com.br/articles/viewcomp.asp?comp=5932&hl=

Listagem 2. Código do botão Delta

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  try
    Delta.Close;
    Delta.Data := ClientDataSet1.Delta;
    Delta.Open;
  except
    MessageDlg('Sem registros no Delta',
      mtWarning, [mbOK], 0);
  end;
end;
```

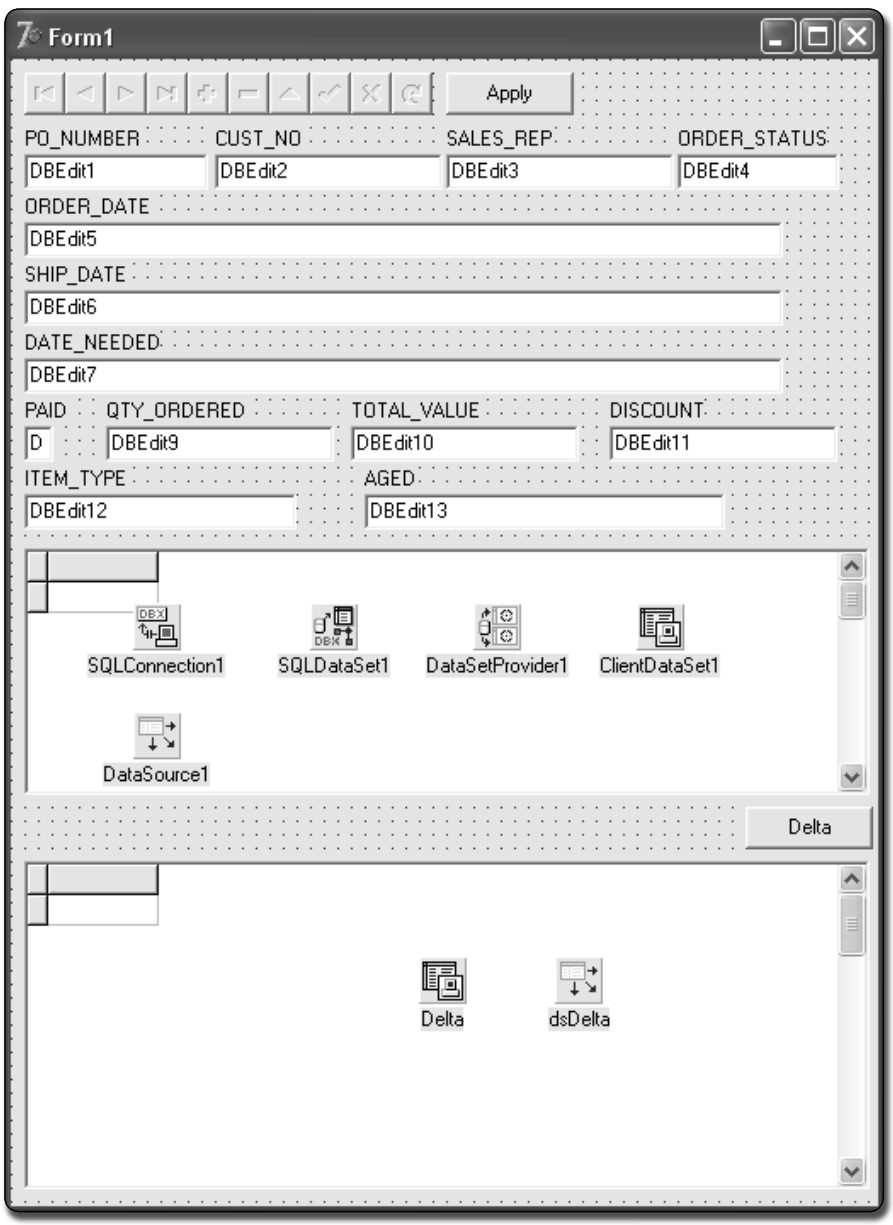


Figura 7. Exemplo de tela com o Delta

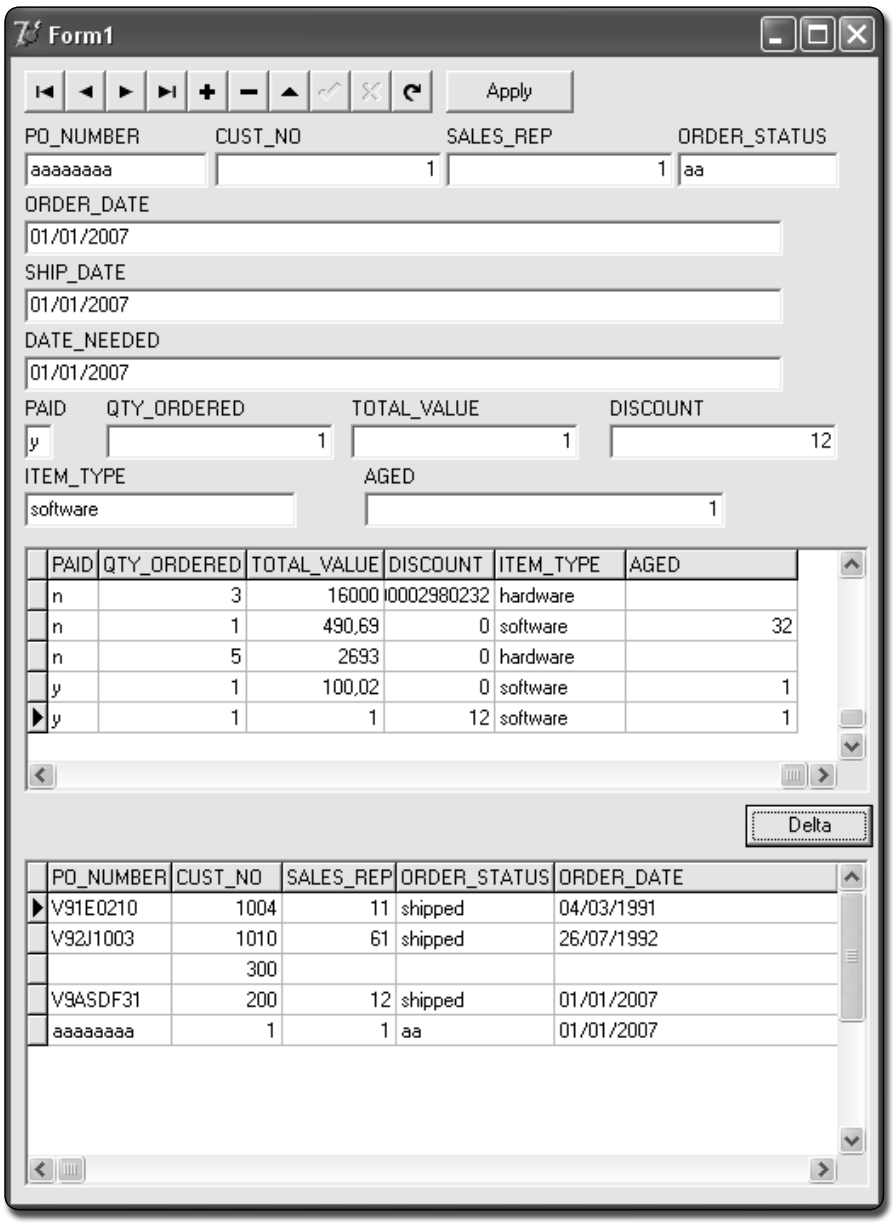


Figura 8. Delta do ClientDataSet1 sendo visualizado

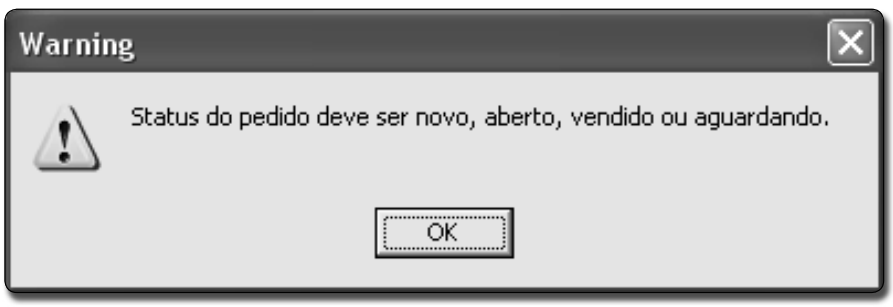


Figura 9. Mensagem personalizada no evento OnUpdateError

O parâmetro de exceção, *E*, possui uma propriedade chamada *OriginalException*. Ele permite que você obtenha o valor original da exceção gerada pelo erro. Com BDE podemos usar a classe *EDBEngineError* para obter o código de erro.

É importante lembrar que dependendo da quantidade de erros mencionada na chamada ao *ApplyUpdates*, as exceções não serão propagadas, isso porque podemos limitar o número de erros tolerados pelo *ClientDataSet*.

Experimente digitar o código da **Listagem 2** no evento *OnUpdateError* do *DSP*, execute a aplicação e digite um valor qualquer no campo Status. Depois salve e clique em *Apply*. Repare na mensagem que é exibida na **Figura 9**.

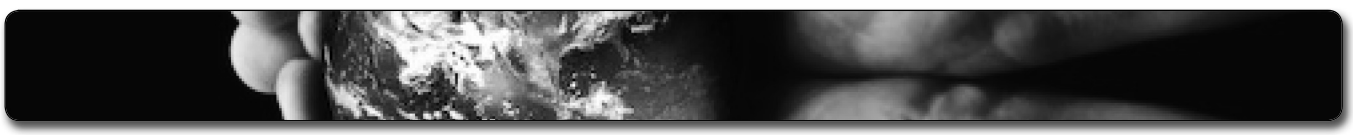
Configurando UpdateMode e ProviderFlags

Trabalhar com *DSP* não é tão difícil quanto se pensa, mas muitos se atrapalham com suas configurações. Uma das coisas que mais sou indagado, diz respeito às propriedades *UpdateMode* do *DataSetProvider* e *ProviderFlags* do *ClientDataSet*. Não há segredos mirabolantes nessas duas propriedades, vejamos o que cada uma significa e como funciona.

Em suma temos apenas três estados do *UpdateMode* que são:

- *upWhereAll:* Utiliza todos os campos na cláusula *Where* para encontrar o registro;
- *upWhereChanged:* Utiliza apenas os campos alterados na cláusula *Where* para encontrar os registros;
- *upWhereKeyOnly:* Utiliza apenas os campos chave para procurar os registros.

Bem, pra ser mais objetivo toda vez que chamamos o método *ApplyUpdates*, o *ClientDataSet* envia os *Data Packet's* para o *DSP* que se encarrega de varrê-lo e montar as instruções SQL que serão enviadas ao servidor e posteriormente aplicadas ao banco. Quando passamos para o *DSP* a opção *upWhereAll*, ele montará uma instrução utilizando todos os campos da tabela em sua cláusula *Where* para que o registro seja



atualizado. Supondo que nossa tabela possui os campos *ID, NOME, ENDERECO, CIDADE, ESTADO* e *TELEFONE*, sendo que apenas o campo *ID* é chave, imagine a instrução montada pelo *DSP* quando apenas o campo *TEFONE* fosse alteração. Veja:

```
UPDATE CLIENTES SET TELEFONE='555-5525'
 WHERE ID=ID, NOME=NOME, ENDERECO=ENDERECO,
 CIDADE=CIDADE, TELEFONE=TELEFONE
```

Perceba que a cláusula *Where* possui todos os campos da tabela, o que é desnecessário. Se o mesmo caso fosse aplicado utilizando a opção *upWhereChanged* a instrução seria:

```
UPDATE CLIENTES SET TELEFONE='555-5525'
 WHERE TEFONE=TELEFONE
```

Nesse caso teríamos problemas, pois poderíamos perder a referência e atualizar o registro incorreto. O melhor de todos os casos é usar a opção *upWhereKeyOnly* que monta a instrução usando apenas os campos da chave primária, ex:

```
UPDATE CLIENTES SET TELEFONE='555-5525'
 WHERE ID=ID
```

Por fim restam as configurações dos campos no *ClientDataSet*. Nesse caso a configuração acontece campo a campo nos *Field's Editor* do objeto. O principal objetivo dos *ProviderFlags* é informar ao *DataSetProvider* qual a providência será tomada com cada campo. O mais importante é entender que nem todos os campos precisam ser alterados no banco de dados, como campos vindos de *Joins* com outras tabelas. Como informar *DSP* que determinado campo pertence a outra tabela e não é necessário gravá-lo? É aí que entram os *ProviderFlags*. Suas opções são:

- *pfInUpdate:* Campo incluso nos *Updates;*
- *pfInWhere:* Campo incluso na cláusula *Where;* Usado para encontrar o registro original;
- *pfInKey:* Campo chave; Usado para encontrar o registro original;
- *pfHidden:* Campo oculto; Campo incluso do *Data Packet*, mas é usado apenas para encontrar o registro original;

Para ser mais direto, cada campo em um *Field's Editor* pode receber um *ProviderFlag* diferente. Então pensamos na seguinte situação: nosso usuário entrou na tela de cadastro de Clientes e alterou apenas o nome do cliente. Ao efetuar um *ApplyUpdates*, o *DSP* entra em ação e fará a montagem da instrução. A instrução mais sensata que o DSP precisa montar consiste apenas em fazer um *Update* usando como campo de alteração o Nome, e na cláusula *Where* necessitamos apenas dos campos da chave, que nesse caso vamos imaginar o *ID* do cliente. Uma instrução seria:

```
UPDATE CLIENTES SET NOME='JOSE DA SILVA'
WHERE ID=ID
```

Para configurar o *ClientDataSet* de tal forma que essa seja a instrução base, precisamos modificar os *ProviderFlags* do campo *ID* parara *[pfInUpdate, pfInWhere, pfInKey]*. Os demais campos ficam configurados apenas com as duas primeiras opções. Dessa forma garantimos que os dados serão atualizados corretamente.

Configurando dinamicamente UpdateMode e ProviderFlags

Essa última etapa de nosso artigo demonstra como fazer a configuração do *UpdateMode* e *ProviderFlags* dinamicamente, o que na verdade não há segredo algum.

A propriedade *UpdateMode* do *DataSetProvider* possui apenas três opções, e para modificá-la em tempo de execução apenas atribua o valor diretamente a sua propriedade, veja:

```
DataSerProvider1.UpdateMode := upWhereAll;
```

Já o *ClientDataSet* a história muda de figura. Nele, sua propriedade *ProviderFlags* é do tipo enumerado, ou seja, pode ter mais de um valor. Mesmo assim ainda é bastante simples de fazer a implementação. Apenas atribua o valor entre colchetes ao campo que deseja adicionar os *ProviderFlags*, como segue:

```
ClientDataSet1.FieldByName('CustNo').
ProviderFlags := [pfInUpdate,pfInKey];
```

Como pode ver, é muito simples. Só é preciso tomar muito cuidado em qual o momento que serão adicionados ou retirados os atributos do *ProviderFlags*, pois toda a regra de negócio ou parte dela, depende dessas propriedades bem configuradas.

Considerações finais

O componente *TDataSetProvider* ainda possui uma série de outras funcionalidades que não puderam ser vistas aqui por conta da sua complexidade, como por exemplo *Nested DataSets* e o uso de pacotes de dados personalizados para montagem de *token's* de validação ou *logs* de evento.

É altamente recomendável fazer um estudo aprofundado de todos os mecanismos e técnicas pra utilização de todos ou pelo menos grande parte dos recursos desse fantástico componente. Importante também lembrar que é perfeitamente possível utilizar outras estruturas de componentes com o DSP, como por exemplo *TDataBases> TQuery >TDataSetProvider>TClientDataSet*.

Conclusão

Falar sobre *TDataSetProvider* não é fácil, visto que sua complexidade é bastante alta e a quantidade de recursos existentes é fantástica. O uso de *DataSetProvider* no dia-a-dia, faz com que a programação seja sempre uma caixinha de surpresa, já que somos capazes de descobrir inúmeras finalidades e funcionalidades para o componente. Por isso é recomendável que se estude a fundo todas as suas particularidades. Nesse artigo vimos os principais aspectos da programação com *DSP* e seus principais recursos, dicas e macetes.

Espero que tenham gostado. Um forte abraço e até a próxima. ●

Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

ClientDataSet

Automatizando o tratamento de Erros

Quem já criou algum tipo de aplicativo que fizesse acesso com banco de dados e que teve que implementar alguma forma para tratar os erros retornados pelo SGBD, sabe o quanto vai ser útil a utilização desse componente (criado neste artigo), pois com ele não será mais necessário ter que implementar algum método de tratamento em todos os componentes ClientDataset espalhados pelo aplicativo.

O principal objetivo desse componente é retirar do desenvolvedor a necessidade de implementar rotinas para tratar os erros retornados no banco de dados através dos eventos OnReconcileError e OnPostError. Nesse artigo irei criar esse novo componente derivado da classe TClientDataset. Os erros tratados pelo componente são baseados nas mensagens retornadas utilizando o banco de dados Firebird.

Rodrigo Lazoti
(rodrigolazoti@yahoo.com.br)
é programador e desenvolvedor Delphi, .Net, Java,Php e Asp. Possui certificação SCJP e atualmente trabalha como consultor J2EE.

Criando o componente MyClientDataset

Vamos iniciar a criação de componente, para isso inicie o Delphi (estou utilizando a versão 7, mas nada impede que se use outra versão), no menu principal, entre na opção Component >> New Component. Com isso será aberta a tela da **Figura 1**. Veja a **Tabela 1**.

Após informar os valores acima, clique em OK, será criado o arquivo fonte do nosso componente. Agora só precisamos implementar as novas funcionalidades no componente. Vamos iniciar incluindo no uses do nosso componente as unit's necessárias para compilação e criar um novo tipo chamado TMyErrors para identificar os erros retornados (**Listagem 1**).

Com isso concluído, vamos inserir os métodos, o construtor e o destrutor do componente, irei criar duas funções sobrecarregadas com o nome de DetectarErros para identificar o erro retornado pelo SGBD, ver em qual opção do nosso

tipo criado a mensagem de erro se enquadra e retorná-la como resultado da função. Criarei também dois procedimentos sobrecarregados com o nome de RetornarErros que tem como finalidade disparar uma nova exceção tratada para o aplicativo. E para finalizar criarei dois métodos para fazer o vínculo com os eventos onReconcileError e onPostError do componente. Primeiro vamos incluir as declarações da sessão private do componente (**Listagem 2**).

Antes de criarmos o corpo desses métodos vamos incluir os métodos restantes para criar o corpo de todos os métodos de uma vez. Veja na **Listagem 3** os métodos da sessão *public*.

Pronto, agora já temos todos os métodos que iremos usar em nosso componente declarados. Aperte Ctrl+Shift+C para que seja criado automaticamente o corpo de todos os métodos do compo-

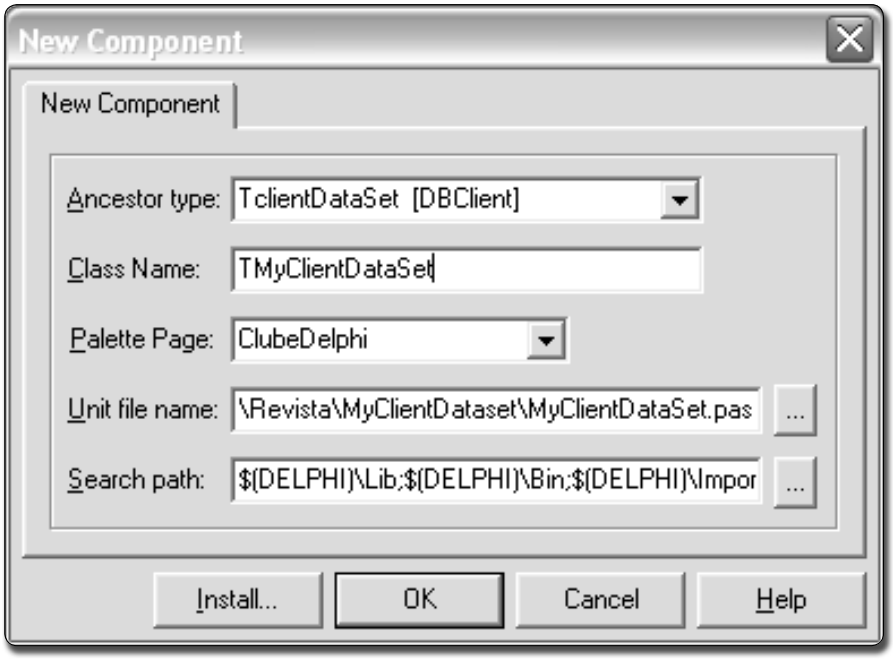


Figura 1. Criando um novo componente derivado da classe TClientDataset

esteja preparado para o futuro!
treinamentos
profissionais em TI
CobiT
ITIL
Material Didático em Português • Turmas Reduzidas • Instrutores Certificados
COMPLETA INFRA-ESTRUTURA • FÁCIL LOCALIZAÇÃO • COFEE-BRAKE • ESTACIONAMENTO CONVENIADO
Faça Tecnologia em Sistema de Informação na FMG: Formamos profissionais para as Maiores Empresas do Brasil
Contribuindo para o aperfeiçoamento profissional dos profissionaios de TI, a **FMG IT Learning**, através de parcerias oficiais, promove os Treinamentos **ITIL Foundations, CobiT, SOX, PMI e ISO.**
O objetivo destes programas de formação é dotar o gestor de conhecimentos implícitos na metodologia das melhores práticas de gestão, com conteúdo e estudos reais de caso. especificamente elaborada para a área de TI.
fmg IT Learning
SOLICITE INFORMAÇÕES SOBRE NOVAS TURMAS
rua general glicério, 45 • 3°andar • cep: 09015-190 • santo andré, são paulo – brasil
Informações: 11 **4427.4687** | www.fmgnet.com.br | cursos@fmgnet.com.br
www.fmgnet.com.br

Campo	Valor	Descrição
Ancestor type	TClientDataset	Contém a classe que nosso componente será derivado.
Class name	TMyClientDataset	O nome da classe do novo componente
Palette Page	ClubeDelphi	O nome da aba onde o componente será instalado.
Unit file name	C:\Arquivos de programas\Borland\Delphi7\Lib\MyclientDataset.pas	Nome e local do arquivo fonte do componente criado.

Tabela 1. Informações iniciais para criação do componente

Listagem 1. Declarando o novo tipo de retorno e incluindo as unit's no uses

```
unit MyClientDataSet;

interface

uses
  SysUtils, Classes, DB, DBClient;

type
  TMyErrors = (meViolacaoChave, meValidacao, meChavePrimaria, meChaveEstrangeira,
meConflito, meOutros);

type
  TMyClientDataSet = class(TClientDataSet)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('ClubeDelphi', [TMyClientDataSet]);
end;

end.
```

Listagem 2. Declarando novos métodos no componente

```
  private
    { Private declarations }
    FTratarErros :Boolean;
    function DetectarErros(e: EReconcileError): TMyErrors; overload;
    function DetectarErros(e: EDatabaseError): TMyErrors; overload;
    procedure RetornarErros(cds: TClientDataSet; Err :EReconcileError); overload;
    procedure RetornarErros(cds: TClientDataSet; Err :EDataBaseError); overload;
```

Listagem 3. Declarando novos métodos no componente

```
    public
    { Public declarations }
    constructor Create(aOwner :TComponent); override;
    destructor Destroy; override;
    procedure MyReconcileError(DataSet: TCustomClientDataSet; E: EReconcileError;
                               UpdateKind: TUpdateKind; var Action: TReconcileAction);
    procedure MyPostError(DataSet: TDataSet; E: EDatabaseError;
     var Action: TDataAction);
```

nente e insira a codificação dos métodos conforme a **Listagem 4**.

O código está todo comentado para fácil entendimento. Pronto, nosso componente está finalizado, salve a unit. No menu principal do Delphi clique em Component >> Install Component. Na opção Unit file name informe o local e nome da unit criada, clique em OK, será aberta uma janela com o pacote DCLUSR.DPK, compile o pacote, com isso o novo componente já estará disponível para o uso. Experimente utilizar o novo componente com os componentes do dbExpress acessando o Firebird, e veja que erros do servidor, como violação de chave, serão automaticamente tratados.

Conclusão

Como vimos nesse artigo, a criação de componentes pode nos ajudar a não escrever códigos repetitivos, tornando a criação a manutenção dos aplicativos mais fácil. ●

Listagem 4. Codificando os métodos do componente

```
constructor TMyClientDataSet.Create(aOwner: TComponent);
begin
  // Associa os eventos aos métodos
   inherited Create(aOwner);
   Self.OnPostError := MyPostError;
   Self.OnReconcileError := MyReconcileError;
end;
destructor TMyClientDataSet.Destroy;
begin
   inherited Destroy;
end;
function TMyClientDataSet.DetectarErros(e: EReconcileError): TMyErrors;
begin
   // Verifica qual o erro ocorrido pela mensagem
   if (Pos('VALIDATION ERROR', UpperCase(E.Message))<>0) then
     Result := meValidacao
   else if (Pos('VIOLATION OF FOREIGN KEY', UpperCase(E.Message))<>0) then
     Result := meChaveEstrangeira
   else if (Pos('VIOLATION OF PRIMARY OR UNIQUE KEY',
     UpperCase(E.Message))<>0) then
     Result := meChavePrimaria
   else if (Pos('RECORD NOT FOUND OR CHANGED BY ANOTHER USER',
     UpperCase(E.Message))<>0) then
     Result := meConflito
   else
     Result := meOutros;
end;
function TMyClientDataSet.DetectarErros(e: EDatabaseError): TMyErrors;
begin
  // Trata Key Violation
   if (Pos('KEY VIOLATION', UpperCase(E.Message))<>0) then
     Result := meViolacaoChave
   else
     Result := meOutros;
end;
procedure TMyClientDataSet.RetornarErros(cds: TClientDataSet;
   Err: EReconcileError);
Begin
   // Faz o tratamento do OnReconcileError
   // Cancela o Update
   cds.CancelUpdates;
   cds.Refresh;
   // Trata o tipo de erro
   case DetectarErros(Err) of
    meValidacao:
    raise Exception.Create('Erro ao salvar ' + cds.Name + '.'+#10#13+
     'Valor inválido ou nulo foi encontrado em campos que contém restrições. '+
     'Motivo: ' + Err.Message);
    meChavePrimaria:
    raise Exception.Create('Erro ao salvar ' + cds.Name + '.'+#10#13+
     'O registro não pode ser gravado, ocorreu um erro de duplicidade na chave primária. '+
     'Motivo: ' + Err.Message);
     meChaveEstrangeira:
      raise Exception.Create('Erro ao efetuar operação ' + cds.Name +
       '.'+#10#13+'Dependência inválida entre tabelas ligadas. ' +
       'Motivo: ' + Err.Message);
     meConflito:
      raise Exception.Create('Erro ao salvar ' + cds.Name + '.'+#10#13+
       'O registro foi modificado ou deletado por outro usuário. '+
       'Motivo: ' + Err.Message);
     else
      raise Exception.Create('Erro ao salvar ' + cds.Name + '.'+#10#13+
       'Erro desconhecido: '+err.Message);
   end;
end;

Procedure TMyClientDataSet.RetornarErros(cds: TClientDataSet;
   Err: EDataBaseError);
Begin
   cds.CancelUpdates;
   cds.Refresh;
   case DetectarErros(Err) of
     meViolacaoChave:
       raise Exception.Create('Erro ao salvar ' +cds.Name+ '.'+
       'Já existe um registro com o código informado!');
     else
      raise Exception.Create('Erro ao salvar ' + cds.Name + '.'+#10#13+
       'Erro desconhecido: '+err.Message);
   end;
end;
procedure TMyClientDataSet.MyPostError(DataSet: TDataSet;
   E: EDatabaseError; var Action: TDataAction);
begin
   RetornarErros(Self, E);
end;
procedure TMyClientDataSet.MyReconcileError(
   DataSet: TCustomClientDataSet;
   E: EReconcileError; UpdateKind: TUpdateKind;
   var Action: TReconcileAction);
begin
   RetornarErros(Self, E);
end;
```

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Controle on-line de vídeo-locadora - Parte 2

Veja como criar um sistema on-line de controle para uma vídeo-locadora

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.

Neste artigo veremos a continuação da criação de um sistema *online* para vídeo-locadora, onde faremos a conclusão dos principais processos que envolvem reservas, retiradas e devolução de filmes por meio de usuários e também a disponibilidade de consultas. Nesta etapa do artigo faremos a criação de páginas para autenticação de usuários e também para autenticação de administradores do sistema de locadora.

Para reservas, criaremos uma página específica aos usuários onde os mesmos poderão realizar buscas de filmes e reservar os mesmos para retirada na locadora. Também a criação de uma área administrativa para efetivação das retiradas feitas por meio de reservas *online* e manutenções de empréstimos, podendo registrar novas locações ou devolvendo filmes locados, centralizando este processo aos administradores da locadora.

Relembrando o artigo anterior, criamos o banco de dados com toda a estrutura de tabelas necessárias, além da configuração da conexão do sistema com o banco de dados utilizando os componentes específicos para conexão com o Firebird. Criamos também no artigo anterior as páginas de manutenção dos gêneros e vídeos da locadora, onde criamos todos os nossos componentes de conexão e pesquisas no código da página (*runtime*), processo este que terá continuação nesta segunda etapa do artigo.

Login de Usuários

Retomando o projeto criado no artigo anterior, após reaberto o mesmo adicione uma nova *ASP.NET Page*, através do menu *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Page* e altere seu nome para "login.aspx". Adicione ao corpo da página uma tabela de 4 linhas e 1 coluna com 300 pixels de largura. Na primeira linha

digite o texto "Login de Clientes" e faça a formatação do texto. Na segunda linha da tabela adicione uma nova tabela com 2 linhas e 2 colunas ocupando 100% da largura disponível, onde deverá digitar nas linhas da primeira coluna o texto "Usuário" e "Senha" e nas linhas da segunda coluna adicionar dois componentes *TextBox(*"txtUser", "txtSenha") alterando a propriedade *TextMode* do *txtSenha* para "*Password*". Adicione na linha 3 um *Button(*"btnLogin") e na última linha da tabela crie um *link* para o *login* dos administradores, direcionando à página *login_adm.aspx* que será criada no decorrer do artigo.

Acessando o código da página (use a tecla de atalho *F12*), crie a constante "strConexao" na área *Interface* da página informando a *string* de conexão com o banco. Esse valor será usado mais adiante para que o componente *fbConnection* acesse o *BD*. Da mesma maneira como utilizado no artigo anterior, declare a constante conforme código a seguir:

```
const
  strConexao = 'User=SYSDBA;
  Password=masterkey;
  Database=<Caminho>LOCADORA.FDB;';
```

De volta ao corpo da página (*F12*), com um duplo clique sobre o *btnLogin*, adicione ao seu evento *OnClick* o código da **Listagem 1** responsável por realizar a autenticação do usuário e senha digitados e direcionar o usuário a página correspondente no sistema. O código utilizado acompanha a metodologia empregada no artigo da edição anterior, onde logo de início criamos todos os componentes e atribuímos a *string* de conexão com o banco ao *Conn*, componente de conexão. Em seguida atribuímos a conexão ao objeto interno do *DataAdapter* chamado *SelectCommand*. É ele que receberá a instrução *SQL* de seleção dos dados em sua propriedade *CommandText*.

Em seguida atribuímos os parâmetros de usuário e senha para a *SQL* de seleção do *CommandText* através do objeto *FbParameter* criado e instanciado separadamente para cada parâmetro adicionado, onde ao final executamos a consulta e relacionamos o resultado a um objeto *FbDataReader* que será verificado quanto ao sucesso do *login*. No caso de informações corretas na autenticação, algumas informações contidas no objeto serão armazenadas em variáveis de sessão (*Session*) para serem usadas futuramente para identificação e validação de usuários. Adicione também junto as *Uses* da página o *namespace FirebirdSQL.Data.Firebird* que possibilitará o trabalho com o objetos de conexão e pesquisa para Firebird.

Validando usuários na página

A validação dos usuários na página é bastante simples de ser implementada. Localize no projeto o *User Control uccontrole.ascx* criado já no início do artigo e com um duplo clique sobre uma área em branco, acesse o evento *Load* do mesmo onde deverá digitar as seguintes linhas de código:

```
try
  LblUsuario.Text := Session['NOME'].
ToString;
except
  Response.Redirect('login.aspx');
end;
```

A existência de um bloco *try..except* é muito importante neste trecho do código para interpretar a seguinte situação:

- Supondo que o usuário esteja logado no sistema, logo a *Session['NOME']* terá um valor relacionado e o mesmo será exibido no *LblUsuario*, mas no caso de o usuário não ter realizado a autenticação e a seção *Nome* não existir, um tratamento de erros faz com que o bloco *except* redirecione o usuário de volta à página de *login*, inibindo seu acesso a todas as páginas onde o *User Control* estiver sendo utilizado.

Página de Reservas OnLine do cliente

No menu *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Page* crie uma nova página salvando-a como "user_reservas.aspx". Adicione ao corpo da página uma tabela de 11 linhas e 1 coluna com 700 pixels de largura. Adicione alguns compo-

Listagem 1. Evento Click para autenticação de usuário e senha dos clientes

```
procedure TWebForm1.btnLogin_Click(sender:
  System.Object; e: System.EventArgs);

var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Conn: FbConnection;
  prUser : FbParameter;
  prSenha : FbParameter;
  fbReader : FbDataReader;

begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'select clientes.nome, clientes.cod_cliente ' +
    'from clientes where ((clientes.login = ?) and
    (clientes.senha = ?))';
  prUser := FbParameter.Create;
  prSenha := FbParameter.Create;
  DataAdapter.SelectCommand.Parameters.Add(prUser);
  DataAdapter.SelectCommand.Parameters.Add(prSenha);
  DataAdapter.SelectCommand.Parameters[0].Value :=
    txtUser.Text;
  DataAdapter.SelectCommand.Parameters[1].Value :=
    txtSenha.Text;
  fbReader := DataAdapter.SelectCommand.ExecuteReader;
  if fbReader.Read then

  begin
    Session['USUARIO'] := txtUser.Text;
    Session['NOME']    := fbReader['NOME'].ToString;
    Session['CODIGO']  := fbReader['COD_CLIENTE']
      .ToString;
    Response.Redirect('user_reservas.aspx');
  end else
  RegisterStartupScript('erro','<script>javascript
    :alert(''Usuário ou Senha incorretos!'');
    </script>');
end;
```

nentes ao corpo dela, organizando-os respectivamente de acordo com o *layout* exibido na **Figura 1.** Arraste na primeira linha da tabela o *User Control uccontrole.ascx*, após insira um *Button(*"btnSair") na segunda linha, seguido de uma linha com o texto "Minhas Locações Ativas" identificando o conteúdo a ser carregado no *DataGrid*("gridLocacao") da linha seguinte. Uma linha em branco com um espaço irá separar o conteúdo do texto "Minhas Reservas Ativas" identificando o *DataGrid*("gridReservas") que será preenchido com as reservas ativas do usuário. As últimas da tabela serão caracterizadas por cadastrar novas reservas, onde além do texto informativo "Criar nova reserva" um componente *DropDowList(*"ddlVideos") para listagem dos vídeos disponíveis para reserva e um *Button(*"btnReservar") para salvar o processo da reserva.

Selecionando o *gridLocacao*, faremos agora algumas configurações necessárias antes da codificação dos processos que irão exibir as informações na página. Com o botão direito do mouse sobre o *DataGrid* acesse sua propriedade *Auto Format* e aplique um estilo de formatação desejado. Novamente com o botão direito do mouse, acesse agora o item *Property Builder* onde iremos modificar a estrutura da categoria *Columns*. Desmarque o item *Create columns automatically at run time*. Em seguida clique no item *Bound Column* em *Available columns* e envie-o para *Selected columns*. Agora selecione-o e atribua o valor *COD_LOCACAO* na propriedade *Data Field*, onde receberemos o valor do campo na tabela *Locacao* através do *result set* da *Select* que faremos nos registros com *status* "N" indicando Locação. Desmarque também a opção *Visible* da coluna.

Agora insira uma nova coluna, ou seja, clique novamente em *Bound Column* e envie-o para a área *Selected columns*. Na propriedade *Header text* ("Texto de cabeçalho"), digite "Titulo" e em *Data Field* o valor *TITULO* que é justamente o campo resultante de um *Inner Join* da tabela *Locacao* com a tabela *Videos* semelhante como no passo anterior. Por fim, adicione uma nova coluna e altere sua propriedade *Header text* para "Prev. Dev". em *Data Field* digite o valor *PREV_DEVOLUCAO* e em *Data Formatting Expression* o valor "{0:dd/MM/yyyy}" que fará a formatação da forma de exibição da data.

Selecionando o *gridReservas*, faremos configurações semelhantes as aplicadas na configuração do *gridLocacao*. Com o botão direito do mouse sobre o *DataGrid* acesse sua propriedade *Auto Format* e aplique um estilo de formatação ao mesmo. Novamente com o botão direito do mouse, acesse agora o item *Property Builder* onde iremos modificar a estrutura da categoria *Columns*, onde no topo da janela desmarque o item *Create columns automatically at run time*. Em seguida clique no item *Bound Column* em *Available columns* e envie-o para *Selected columns*. Agora selecione-o e atribua o valor *COD_LOCACAO* na propriedade *Data Field*, onde receberemos o valor do campo na tabela *Locacao* através do *result set* da *Select* que faremos nos registros com *Status* "R" para Reservas. Desmarque também a opção *Visible* da coluna.

Agora insira uma nova coluna, ou seja, clique novamente em *Bound Column* e envie-o para a área *Selected columns*. Na propriedade *Header text* ("Texto de cabeçalho"), digite "Titulo" e em *Data Field* o valor TITULO que é justamente o campo resultante de um *Inner Join* da tabela *Locacao* com a tabela *Videos* semelhante como no passo anterior. Por fim, adicione uma nova coluna e altere sua propriedade *Header text* para "Data Reserva". em *Data Field* digite o valor "DATA" e em *Data Formatting Expression* o valor "{0:dd/MM/yyyy}" que fará a formatação da forma de exibição da data.

Selecionando o *ddlVideos*, adicione a sua propriedade *DataValueField* o valor

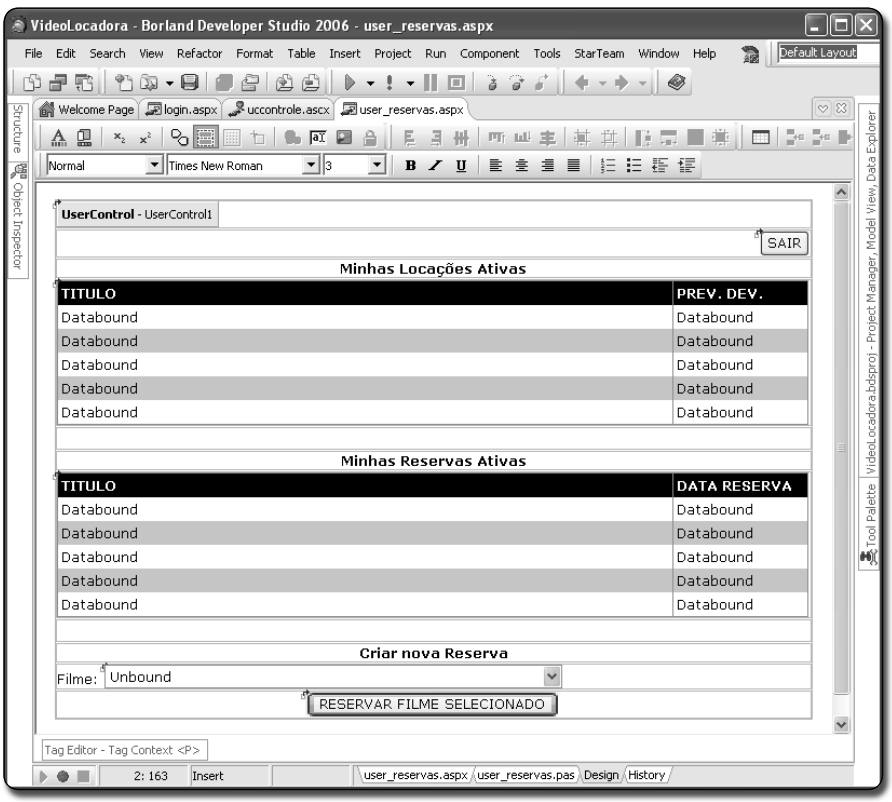


Figura 1. Layout da página de reservas do usuário

Nota do DevMan

Para alterar a largura das colunas, basta entrar novamente no Property Builder e selecionar o item Format. Nele podemos definir atributos para várias características do componente Data Grid. Note que temos a opção Columns. Expanda este item e clique na última coluna ("Alterar"). À direita digite 50 no campo Width ("largura") e confirme.

COD_VIDEO e para *DataTextField* o valor *TITULO*, sendo estas duas colunas de informações que serão carregadas através do *result* de um *select* em todos os vídeos disponíveis para reserva.

O processo para carregamento das informações requer que 1 *procedure* e uma *function* sejam criadas, especificando para cada uma delas componentes que serão criados em tempo de execução conforme exemplos anteriores e que requerem a criação de uma constante "strConexao" e inclusão da *namespace FirebirdSQL.Data.Firebird* na cláusula *Uses*. Declare na seção *Private* do código a *procedure* e também a *function* conforme lista adiante e adicione a cada uma delas os respectivos códigos encontrados na **Listagem 2**, sendo que todos os processos encontram-se comentados entre as linhas do mesmo. Observe que para carregar as locações e reservas ativas utilizamos apenas uma função, a qual requer a passagem do parâmetro *Status* de acordo com a situação cadastral dos registros que desejamos exibir:

```
procedure CarregaVideos;
function CriaListaVideos(Status:
  String):DataSet;
```

Finalizada a criação e codificação dos métodos para carregar os dados de locações, reservas e vídeos precisamos fazer a chamada dos mesmos. Para isso usaremos o evento *Load* da página onde deverá digitar as seguintes linhas do código, que irão jogar o resultado aos componentes relacionados. Observe que estamos utilizando a mesma função para carregar as reservas e locações, alterando apenas o valor do parâmetro *Status* repassada para a function:

```
gridLocacao.DataSource :=
CriaListaVideos(''N'');
gridLocacao.DataBind;
gridReservas.DataSource :=
CriaListaVideos(''R'') AND
  (LOCACAO.PREV_DEVOLUCAO >= ''TODAY'');
gridReservas.DataBind;
CarregaVideos;
```

Se executar a página neste momento, já poderá perceber que todas as informações serão listadas, porém ainda está disponível o cadastro de novas reservas. Uma nova reserva será realizada a partir do *click* ao botão *btnReservar* no qual adicione o código da **Listagem 3**.

Listagem 2. Codificando as ações para carregamento de vídeos, reservas e locações

```
procedure TWebForm2.CarregaVideos;
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Ds: DataSet;
  Conn: FbConnection;

begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura
    do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'SELECT V.COD_VIDEO, V.TITULO FROM VIDEOS V '+
    'WHERE V.COD_VIDEO NOT IN (SELECT LOCACAO.VIDEO
      FROM LOCACAO '+
    'WHERE ((LOCACAO.VIDEO = V.COD_VIDEO) AND
    ((LOCACAO.STATUS = ''R'') '+
    'OR (LOCACAO.STATUS = ''N'')))) ORDER BY V.TITULO';
  { Criação em memória do DataSet auxliar }
  Ds := DataSet.Create;
  DataAdapter.Fill(Ds, 'Titulo');

  Try
    { Popularização dos vídeos no componente da
      página }
    ddlVideos.DataSource := Ds;
    ddlVideos.DataBind;
  finally
    Conn.Close;
  end;
end;

function TWebForm2.CriaListaVideos(Status: String):
  DataSet;
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Ds: DataSet;
  Conn: FbConnection;
  prCliente : FbParameter;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText :=
    'SELECT LOCACAO.COD_LOCACAO, VIDEOS.TITULO,
    '  CLIENTES.NOME, ' +
    'LOCACAO.PREV_DEVOLUCAO, LOCACAO.DATA FROM  ' +
    '  LOCACAO ' +
    '   INNER JOIN VIDEOS ON (LOCACAO.VIDEO =  ' +
    '  VIDEOS.COD_VIDEO) ' +
    '   INNER JOIN CLIENTES ON (LOCACAO.CLIENTE = ' +
    '  CLIENTES.COD_CLIENTE) ' +
    'WHERE ((LOCACAO.STATUS = ' + Status + ')
    ' AND (LOCACAO.CLIENTE = ?)) ORDER BY  ' +
    ' LOCACAO.DATA';
  { Criação dos parâmetros da pesquisa. O Status
    será configurado ao chamar a function }
  prCliente := FbParameter.Create;
  DataAdapter.SelectCommand.Parameters.Add(
    prCliente);
  DataAdapter.SelectCommand.Parameters[0].Value :=
    Session['CODIGO'].ToString;
  { Criação em memória do DataSet auxliar }
  Ds := DataSet.Create;
  DataAdapter.Fill(Ds, 'Titulo');

  try
    { Joga o resultado do Select para o Result
      da função }
    Result := Ds;
  finally
    Conn.Close;
  end;
end;
```

Listagem 3. Código para reservar um vídeo

```
procedure TWebForm2.btnReservar_Click(sender:
  System.Object; e: System.EventArgs);

var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Conn: FbConnection;
  prVideo : FbParameter;
  prCliente : FbParameter;
  prData : FbParameter;
  prStatus : FbParameter;
  prDev : FbParameter;

begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;

  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;

  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.InsertCommand := Comand;
  DataAdapter.InsertCommand.Connection := Conn;
  DataAdapter.InsertCommand.CommandText :=
    'INSERT INTO LOCACAO '+
    '(VIDEO, CLIENTE, DATA, STATUS, PREV_DEVOLUCAO)
    'VALUES (?, ?, ?, ?, ?)';

  prVideo   := FbParameter.Create;
  prCliente := FbParameter.Create;
  prData    := FbParameter.Create;
  prStatus  := FbParameter.Create;
  prDev     := FbParameter.Create;
  DataAdapter.InsertCommand.Parameters.Add(prVideo);
  DataAdapter.InsertCommand.Parameters.Add(prCliente);
  DataAdapter.InsertCommand.Parameters.Add(prData);
  DataAdapter.InsertCommand.Parameters.Add(prStatus);
  DataAdapter.InsertCommand.Parameters.Add(prDev);
  DataAdapter.InsertCommand.Parameters[0].Value :=
    ddlVideos.SelectedValue;
  DataAdapter.InsertCommand.Parameters[1].Value :=
    Session['CODIGO'].ToString;
  DataAdapter.InsertCommand.Parameters[2].Value :=
    DateTime.Today.ToString('dd.MM.yyyy');
  DataAdapter.InsertCommand.Parameters[3].Value := 'R';
  DataAdapter.InsertCommand.Parameters[4].Value :=
    System.&String.Format('{0:dd/MM/yyyy}',
    IncDay(DateTime.Today, 7));{Uses DateUtils}

  if DataAdapter.InsertCommand.ExecuteNonQuery> 0 then
  begin
    gridReservas.DataSource := CriaListaVideos(''R'')
      AND (LOCACAO.PREV_DEVOLUCAO >= ''TODAY''');
    gridReservas.DataBind;
    CarregaVideos;
  end;
end;
```

O código também se utiliza da criação dos componentes em runtime onde passamos uma instrução SQL ao objeto InsertCommand do DataAdapter e configuração dos parâmetros de cadastros. Ao executarmos a instrução, e o resultado for de sucesso, chamamos os métodos para que a página seja recarregada e as informações atualizadas. Lembre-se de adicionar nas Uses da página o namespace DateUtils da qual estamos chamando uma função para incrementar certa quantidade de dias à data de vencimento da reserva.

Adicione também ao evento *OnClick* do *btnSair* o código adiante para fazer o redirecioamento da página e após execute a página fazendo uma nova reserva para o usuário autenticado.

```
Response.Redirect('login.aspx');
```

Conclusão

Não perca a próxima parte deste artigo, onde continuaremos a implementação das retiradas e devolução de filmes por meio de usuários e também a disponibilidade de consultas. ●

Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback

Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **2000 vídeo-aulas** d site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

Todas as vídeos que você sempre quis ver a partir de **R$2,50!**

Saiba mais sobre o Sistema de Créditos!

DevMedia

www.devmedia.com.br/creditos

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Desenvolvendo uma Aplicação Completa com PocketStudio

Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 3

Ricardo C. Boaro
(rboaro@aquasoft.com.br)
trabalha com desenvolvimento de sistemas em Delphi há mais de 10 anos e PocketStudio há 3 anos. Atualmente é gerente de informática na Di Hellen Indústria de Cosméticos, e atual como instrutor certificado Borland na Aquasoft Tecnologia da Informação parceira da Borland, em Porto Alegre – RS. Borland Instrutor, Delphi 7, 2007 e Certified.

No artigo anterior, aprendemos a criar uma tabela e fazer referência a ela em nosso formulário de consulta de clientes. Com base no mesmo conceito e aprendizado, faremos a inclusão de novas telas em nosso sistema, tais como: cadastro de produtos e digitação de pedidos.

Lembrando que nossas telas no sistema apenas consultam a base de dados não possibilitando a inclusão de registros. Essa tarefa será efetuada através do sincronismo entre dispositivo e PC.

Poderíamos criar e utilizar a tela para editar os registros e enviá-los ao PC, porém em nosso exemplo será apenas mais uma janela de consulta. As telas que irão manipular informações são as de pedidos e itens do pedido.

O primeiro passo é criarmos a tabela produtos para armazenarmos os dados e podermos navegar entre os registros. Lembrando que essa tabela será preenchida no momento do sincronismo com o PC. Para isso devemos criar uma nova *Unit*, como vimos no artigo anterior. Acesse o menu *File|New>Unit*, salve-a com o nome de "ProdutosDB.pas". Vale lembrar que podemos utilizar as teclas de atalhos *CTRL + Shift + S* para salvar nosso projeto. Feito isso vamos montar a tabela de produtos.

Logo abaixo da palavra reservada *Interface* declare uma seção *Uses* para adicionarmos a *PSL* que é a *Library* do PocketStudio onde estão os métodos necessários para manipularmos o BD dentro da *Unit PSDatabase*. Após a seção *Uses* declaramos uma nova seção *Const* onde informaremos o *DBName*, *DBType* e os índices para recuperação e configuração dos valores nos campos da tabela de *Produtos*.

Após a definição dos campos e seus respectivos índices, declaramos uma seção *Var*, onde informaremos ao PocketStudio o *FieldDefs* e um array. Também iremos declarar as variáveis globais que

irão referenciar nossa base de dados em todo projeto. A estrutura de todas as tabelas criadas no PocketStudio é sempre a mesma, como uma receita de bolo. Mudamos apenas nome da tabela e campos, mas a maneira de montar e manipular é exatamente igual para todas.

Como no artigo anterior criamos a tabela de clientes, não vou entrar em detalhes de como funciona cada função criada em nossa tabela. Altere a *Unit ProdutosDB* conforme a **Listagem 1.**

Criando a Tela de Consulta a Produtos

Criada a tabela de produtos, vamos adicionar um novo formulário para criarmos a tela de consulta. No menu principal do PocketStudio acesse *File|New>Form* e salve-o como "UDadosProdutos.pas" usando *File>Save* ou pelas teclas de atalho *CTRL + S*. Altere a propriedade Name para "FrmDadosProdutos" e seus Title para "Consulta a Produtos". Desenhe uma tela semelhante a **Figura 1**.

Nomeie os componentes do tipo *Field* com os nomes "FldCodigo", "FldDescricao", "FldPreco", "FldEstoque", "FldTamanho" e "FldUn", respectivamente. Insira cinco botões, sendo, quatro na parte inferior que se chamarão "BtnPrimeiro", "BtnAnterior", "BtnProximo" e "BtnUltimo", seguindo da esquerda para a direita. Aproveite e repita os passos do artigo anterior para inclusão de *Bitmap's* e associação deles aos botões da parte inferior. O botão que ficará ao lado do título do formulário se chamará "BtnVoltar".

Antes de codificarmos os botões, precisamos criar uma função para carregar os dados na tela de consulta conforme navegarmos entre os registros. Para isso utilizaremos as funções da *PSDataBase* da *PSL*. Pressione *F12* no formulário para visualizar a página de código e localize a palavra reservada *Implementation*. Declare o procedimento da **Listagem 2** abaixo de *Implementation*.

Figura 1. Exemplo de tela de Consulta a Produtos

Listagem 1. Código completo da tabela de Produtos

```
unit ProdutosDB;
interface
uses PSL;
const
  ProdutosDBName = 'ProdutosDB';
  ProdutosDBType = Rsc('DBPR');
  Prod_Codigo = 0;
  Prod_DescProd = 1;
  Prod_Preco = 2;
  Prod_Estoque = 3;
  Prod_Tam = 4;
  Prod_Un = 5;
var
  FieldDefs : array[0..5] of TFieldDef =
    ((DataType: ftUInt16),
    (DataType: ftString),
    (DataType: ftDouble),
    (DataType: ftUInt32),
    (DataType: ftString),
    (DataType: ftString));
  bProdutosInclui : Boolean;
  DBPro : TDatabase;

function Open: Boolean;
function Close: Boolean;
function ProcuraCodigo(Codigo: UInt16): Boolean;

implementation

function Open: Boolean;
var
  Atributos : UInt16;
begin
  Result := PSDatabase.Open(DBPro, ProdutosDBName,
    dmModeReadWrite);
  if not Result then
  begin
    Result := PSDatabase.CreateDatabase(ProdutosDBName,
      Creator, ProdutosDBType);
    if Result then
      Result := PSDatabase.Open(DBPro, ProdutosDBName,
      dmModeReadWrite);
  end;
  if not Result then
  begin
    ShowSystemError(PSDatabase.LastError);
    Exit;
  end;
  PsDataBase.DataBaseAttributes(ProdutosDBName,
    Atributos);
  Atributos := Atributos and $FFF7;
  PsDataBase.SetDataBaseAttributes(ProdutosDBName,
    Atributos);
  PSDatabase.SetFieldDefs(DBPro, FieldDefs[0],
    SizeOf(FieldDefs) div SizeOf(FieldDefs[0]));
end;
function Close: Boolean;
begin
  Result := PSDatabase.Close(DBPro);
end;

function ProcuraCodigo(Codigo: UInt16): Boolean;
begin
  PSDatabase.First(DBPro);
  while not PSDatabase.EOF(DBPro) do
  begin
    if PSDatabase.FieldUInt16(DBPro, Prod_Codigo) =
      Codigo then
    begin
      Result := True;
      exit;
    end;
    PSDatabase.Next(DBPro);
  end;
  Result := False;
end;
end.
```

Na função *CarregaProdutos* utilizamos uma variável *Buffer* que é um *Array* de *Char* para auxiliar a transformação do código do produto de *UInt16* para *String*. Para isso utilizamos a função *StrIToA*. Após isso recuperamos o valor do campo código da tabela e o transformarmos *String*. Por fim recebemos o valor convertido na propriedade *Text* do *Field*. Para isso utilizamos a *Unit PsField* da *PSL* e o método *SetText*. Um passo importante na função CarregaProdutos é fazer um teste para saber se existem registros na tabela, pois caso não existam, uma exceção será gerada. Para isso usamos o método *RecourdCount* da *Unit PsDataBase*.

O próximo passo é codificarmos os botões, por isso iniciaremos pelo botão de *Voltar*. Dê um clique duplo no botão *BtnVoltar* e seremos levados ao fonte da *Unit* no evento *Select*. Digite apenas o código que segue:

```
FrmGotoForm(FrmPrincipal);
```

Para que possamos fazer referência ao formulário principal e às funções de nossa tabela de produtos declare as *Units ProdutosDB* e *UPrincipal* no *Uses* do formulário atual. Observe o código a serguir:

```
implementation
uses
  ProdutosDB, UPrincipal;
```

Em seguida podemos codificar o restante dos botões. Na **Listagem 3** encontramos o código de cada botão. Observe que a codificação é muito simples. Apenas chamamos os métodos *First* ("primeiro"), *Prior* ("anterior"), *Next* ("próximo") e *Last* ("último"), apropriado para cada botão.

Note que em todos os botões utilizamos a *PsDataBase* e os métodos de navegação passando como parâmetro a variável *DBProd*. Essa variável faz referência a nossa tabela de produtos.

Para finalizar a codificação da tela de consulta de produtos vamos codificar o evento *OnOpen* do formulário para apresentarmos os dados da tabela. Clique no formulário e pressione *F11* para visualizar suas propriedades e eventos. Na aba Events clique duas vezes sobre o evento *OnOpen* e acrescente uma chamada ao procedimento *CarregaProdutos*. O código completo ficará como a seguir:

```
procedure FrmDadosProdutosOpen;
begin
 CarregaProdutos;
 PSForm.Draw;
end;
```

O único detalhe agora é que devemos incluir uma chamada ao formulário de produtos na tela principal de nosso sistema, por isso volte ao formulário principal, clique duas vezes no botão *Produtos* e digite a chamada como segue:

```
FrmGotoForm(FrmDadosCliente);
```

Há mais uma configuração necessária a ser feita em nosso projeto para que os formulários sejam efetivamente chamados. Assim como fizemos no artigo anterior, teremos que alterar o método *ApplicationHandleEvent* de forma que possa chamar o formulário que desejamos. Para isso, abra o arquivo *Vendas* equivalente ao *Project>Source* do Delphi. Pressione *CTRL + F12* e selecione-o. Será necessário fazer a alteração apenas do *case..of* ao final do procedimento. Veja a seguir:

Nota do DevMan

Nas versões mais antigas do PocketStudio, principalmente as versões Trial, a Unit LibAll, que possue diversas funções de conversão, não vinha instalada no diretório de instalação do PS. Por isso, caso tenha problemas ao compilar a aplicação, é recomendado que salve o arquivo LibAll.pas no diretório do seu código fonte. Esse arquivo pode ser encontrado juntamente com os fontes de exemplo desse artigo. Acesse o portal DevMedia e localize a página de downloads das revistas ClubeDelphi.

Listagem 2. Procedimento de carga de Produtos

```
procedure CarregaProdutos;
var
 Buffer: Array [0..10] of Char;
begin
  if PsDataBase.RecordCount(DBPro) > 0 then
  begin
    StrIToA(Buffer, PsDataBase.FieldUInt16(DbPro,
     Prod_Codigo));
    PsField.SetText(FldCodigo, Buffer);

    PsField.SetText(FldDescricao, PsDatabase.
      FieldStringPtr(DbPro, Prod_DescProd));

    FormatFloat(Buffer, PSDatabase.FieldDouble(DBPro,
      Prod_Preco), 2);
    PsField.SetText(FldPreco, Buffer);

    StrIToA(Buffer, PsDataBase.FieldUInt32(DbPro,
      Prod_Estoque));
    PsField.SetText(FldEstoque, Buffer);

    PsField.SetText(FldUn, PsDatabase.FieldStringPtr(
      DbPro, Prod_Un));
    PsField.SetText(FldTamanho, PsDatabase.
      FieldStringPtr(DbPro, Prod_Tam));
  end;
end;
```

Listagem 3. Código dos botões de navegação

```
procedure BtnPrimeiroSelect;
begin
  PsDataBase.First(DBPro);
  CarregaProdutos;
end;

procedure BtnAnteriorSelect;
begin
  PsDataBase.Prior(DBPro);
  CarregaProdutos;
end;

procedure BtnProximoSelect;
begin
  PsDataBase.Next(DBPro);
  CarregaProdutos;
end;

procedure BtnUltimoSelect;
begin
  PsDataBase.Last(DBPro);
  CarregaProdutos;
end;
```

```
case FormID of
  FrmPrincipal:
    FrmSetEventHandler(Form, uPrincipal.
    HandleEvent);
  FrmClientes:
    FrmSetEventHandler(Form,UDadosClientes.
    HandleEvent);
  FrmDadosProdutos:
    FrmSetEventHandler(Form,
    UDadosProdutos.HandleEvent);
end;
```

Veja que adicionamos o formulário *FrmDadosProdutos* ao *case..of*. Cada formulário que vamos adicionando ao sistema, é necessário fazer alteração nesse evento. Veja o evento completo na **Listagem 4**.

Criando a tela de Pedidos e Itens

O primeiro passo antes de criarmos a tela propriamente dita, é criar as tabelas necessárias para armazenar os dados digitados. Uma boa dica, é seguir os mesmos passos usados para criar a tabelas de *clientes* e *produtos*. Vejamos os principais passos a executar:

• Crie uma nova Unit usando o menu File|New>Unit e salve-a com o nome que preferir. Nessa série de artigos estamos usando "NomeDaTabelaDB.pas", ex: ProdutosDB.pas;

Nota do DevMan

Formulários Destrutivos e Não Destrutivos

Até agora trabalhamos apenas com formulários Destrutivos, mas existem duas formas de se trabalhar com formulários no PocketStudio como podemos ver a seguir:

Modo Destrutivo: neste modo todos os objetos visuais do formulário atual e seu conteúdo serão perdidos. Se necessitarmos preservar os dados informados no formulário precisamos criar funções para salvar e restaurar os dados. Nesse modo utilizamos a função FrmGotoForm para alternar entre os formulários.

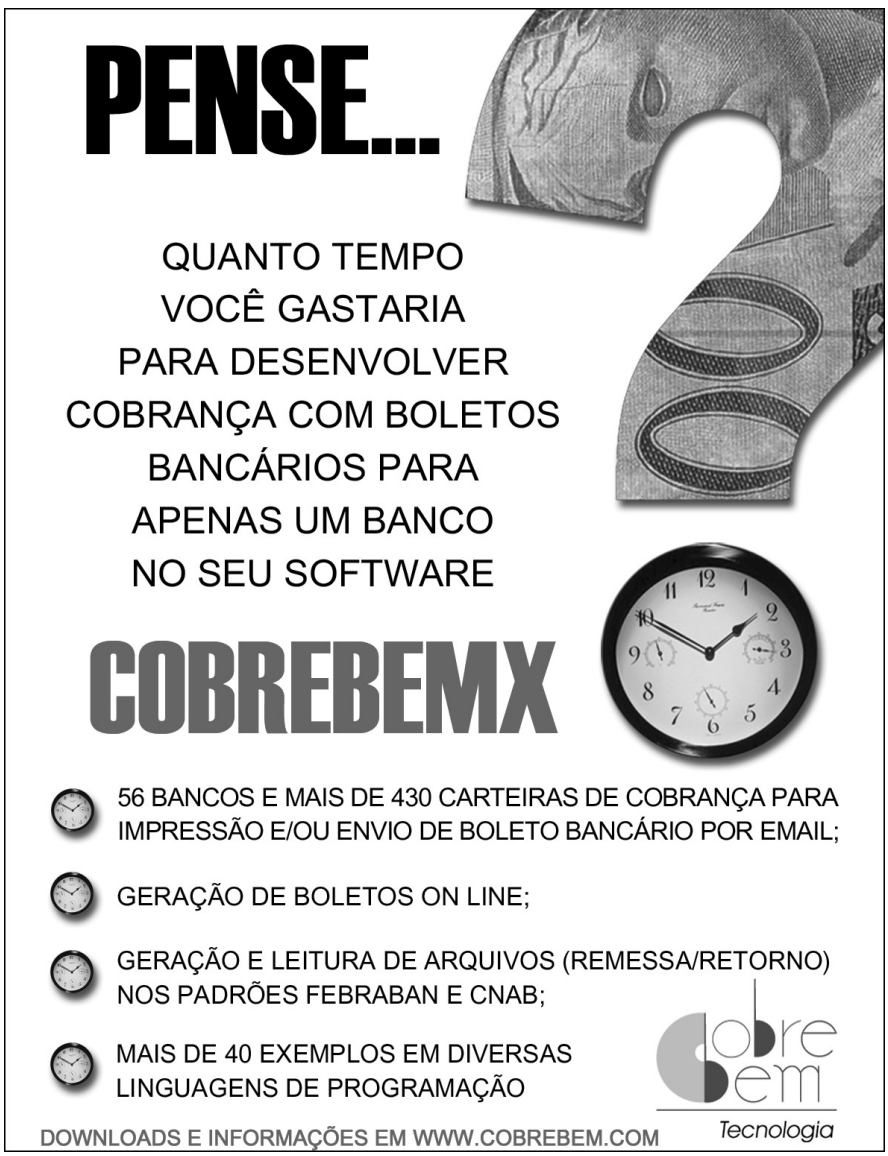
Modo Não Destrutivo: o problema dessa abordagem é que não podemos dar mais de um passo, ou seja, imaginemos três formulários. Podemos navegar do primeiro para o segundo e voltar para o primeiro, mas se tentarmos navegar do primeiro direto para o terceiro formulário, uma exceção será gerada. Nessa abordagem usamos a função FrmPopupForm para ir para o formulário e FrmReturnToForm para retornar.

Tenha em mente que se precisarmos navegar entre vários formulários e voltar aleatoriamente entre eles, devemos utilizar o modo destrutivo. Também devemos considerar o que o modo Não Destrutivo irá carregar a memória do dispositivo à medida que os formulários são chamados.

Em resumo, precisamos considerar cada situação e analisar o que realmente é necessário, evitando maiores problemas na aplicação.

Listagem 4. Código completo do evento ApplicationHandleEvent

```
function ApplicationHandleEvent(var Event: EventType)
  : Boolean;
var
  FormID: UInt16;
  Form: FormPtr;
begin
  Result := False;
  if Event.eType = frmLoadEvent then
  begin
    FormID := Event.frmLoad.formID;
    Form := FrmInitForm(FormID);
    FrmSetActiveForm(Form);
    case FormID of
      FrmPrincipal: FrmSetEventHandler(Form,
        uPrincipal.HandleEvent);
      FrmClientes: FrmSetEventHandler(Form,
        UDadosClientes.HandleEvent);
      FrmDadosProdutos: FrmSetEventHandler(Form,
        UDadosProdutos.HandleEvent);
    end;
    Result := True;
  end
end;
```

• Crie as constantes de nome de campos e as funções necessárias para abrir e fechar a tabela;

• Crie uma função para naveção dos dados, *CarregaDados* por exemplo;

No caso de *Pedidos* e *Itens de Pedidos*, devemos criar duas novas Units, que se chamarão "PedidosDB.pas" e "ItemPedDB.pas". O código fonte completo da criação de ambas tabelas podemos ver nas **Listagens 5** e **6**.

Desenhando a tela de pedidos

A tela de pedidos e itens de pedido fugirá um pouco do que já vimos até agora. Crie um novo formulário usando o menu *File|New>Form* e salve-o como "UPedidos.pas" e seu *Name* modifique para "FrmPedidos". Modifique também a propriedade *Title* para "Cabeçalho do Pedido". Desenhe uma tela semelhante a **Figura 2**.

Há três novidades aqui, ou seja, três novos componentes que não vimos até agora. São eles, *PopupTrigger, SelectorTrigger* e *List*. O componente *PopupTrigger* em conjunto com *List* funciona como se fosse um *ComboBox* do Delphi. Isso significa que podemos simular listas suspensas. Para isso insira no formulário um componente *PopupTrigger*("PopCondicoes") da paleta *Form*. Altere o seu *Caption* para "Cond. Pagto", assim enquanto não houver um item selecionado, a indição de "Condições de Pagamento" estará ativa. Inclua agora um componente *List*("LstCondicoes") e insira alguns itens na propriedade *Items* do controle. Em seguida retorne ao componente *PopCondicoes* e selecione o *LstCondicoes* na propriedade *PopupList*. Isso fará com que a lista de condições de pagamento apareça ao clicar no *PopupTrigger*. Para evitar que o *PopCondicoes* apareça constantemente, marque sua propriedade *Visible* com o valor *False*.

Listagem 5. Código da tabela PedidosDB

```
unit PedidosDB;
interface
uses PSL;

const
  PedidosDBName = 'PedidosDB';
  PedidosDBType = Rsc('DBPE');
  Ped_NumeroPedido = 0;
  Ped_CodCli = 1;
  Ped_Emissao = 2;
  Ped_Cond = 4;
  Ped_Total = 5;

var
  FieldDefs : array[0..4] of TFieldDef =
    ((DataType: ftUInt32),
    (DataType: ftUInt16),
    (DataType: ftDateTime),
    (DataType: ftString),
    (DataType: ftDouble));

  bPedidosInclui              : Boolean;
  dPedidoData                 : DateTimeType;
  DBPed                       : TDatabase;

function Open: Boolean;
function Close: Boolean;
function ProcuraCodigo(Codigo: UInt32): Boolean;

implementation

function Open: Boolean;
begin
  Result := PSDatabase.Open(DBPed, PedidosDBName,
    dmModeReadWrite);
  if not Result then
  begin
    Result := PSDatabase.CreateDatabase(PedidosDBName,
      Creator, PedidosDBType);
    if Result then
      Result := PSDatabase.Open(DBPed, PedidosDBName,
      dmModeReadWrite);
  end;

  if not Result then
  begin
    ShowSystemError(PSDatabase.LastError);
    Exit;
  end;

  PSDatabase.SetFieldDefs(DBPed, FieldDefs[0],
    SizeOf(FieldDefs) div SizeOf(FieldDefs[0]));
end;

function Close: Boolean;
begin
  Result := PSDatabase.Close(DBPed);
end;

function ProcuraCodigo(Codigo: UInt32): Boolean;
begin
  PSDatabase.First(DBPed);
  while not PSDatabase.EOF(DBPed) do
  begin
    if PSDatabase.FieldUInt32(DBPed, Ped_NumeroPedido)
      = Codigo then
    begin
      Result := True;
      exit;
    end;
    PSDatabase.Next(DBPed);
  end;
  Result := False;
end;
end.
```


ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Ricardo Boaro que mostra como trabalhar com os componentes SelectorTrigger e List.

www.devmedia.com.br/articles/viewcomp.asp?comp=7606

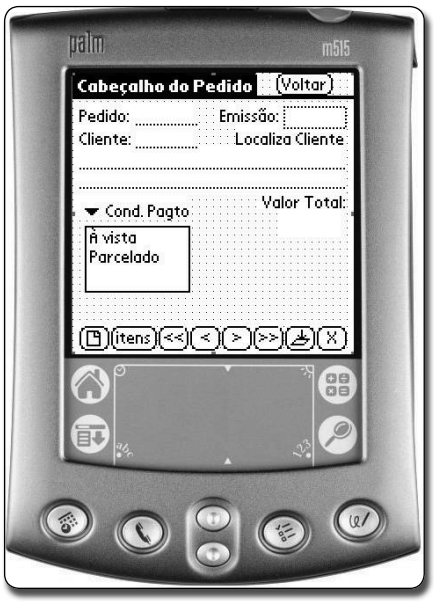


Figura 2. Exemplo de tela de Consulta a Pedidos

Os nomes dos componentes seguem a mesma linha empregada nas telas anteriores. Use o prefixo "Fld" para campos e "Btn" para botões. Nessa janela encontramos os campos "FldNumPedido", "FldCliCodigo", "FldCliNome" e "FldValTotal". E ainda os botões "BtnNovo", "BtnItens", "BtnPrimeiro", "BtnAnterior", "BtnProximo", "BtnUltimo", "BtnFim" e "BtnGravar", respectivamente da esquerda para a direita.

Abaixo do campo *Emissão*, insira um botão comum e troque seu *Caption* para "Localiza Cliente". Aproveite e modifique a propriedade *Frame* usando a opção *frNone*, assim retiramos sua borda.

Com exceção dos controles *SelectorTrigger, PopupTrigger* e *List*, já trabalhamos com os componentes utilizados nessa tela, sendo assim cabe um explicação sobre os novos componentes.

- *SelectorTrigger:* Este componente é da família dos botões, portanto utilizamos a Unit *PsButton* da *PSL* para trabalharmos com ele. A diferença é que podemos chamar formulários em modo modal com ele. Esse componente será usado na *Emissão*, portanto insira um *SelectorTrigger*("SltEmissao") logo à direita do *Label Emissão*;
- *PopupTrigger:* Mais um componente da família dos botões e sua funcionalidade depende do uso em conjunto com um objeto lista. Ele sozinho não tem utilidade, imaginemos ele como o conhecido ComboBox do Delphi, em *run-time* ele é muito similar;
- *List:* Lista é um objeto bastante utilizado por nós desenvolvedores, pois está disponível em praticamente todos os ambientes de desenvolvimento. Durante o desenvolvimento da nossa série

Listagem 6. Código da tabela ItemPedDB

```
unit ItemPedDB;
interface
uses PSL;

const
  ItemPedDBName = 'ItemPedDB';
  ItemPedDBType = Rsc('DBIP');
  Itp_NumeroPed = 0;
  Itp_CodProd = 2;
  Itp_Qtde = 3;
  Itp_Preco = 4;

var
  FieldDefs : array[0..3] of TFieldDef =
    ((DataType: ftUInt32),
    (DataType: ftUInt16),
    (DataType: ftUInt16),
    (DataType: ftDouble));

  bItemPedInclui              : Boolean;
  DBItp                       : TDatabase;

function Open: Boolean;
function Close: Boolean;
function ProcuraCodigo(Codigo: UInt16): Boolean;

implementation

function Open: Boolean;
begin
  Result := PSDatabase.Open(DBItp, ItemPedDBName, dmModeReadWrite);
  if not Result then
  begin
    Result := PSDatabase.CreateDatabase(ItemPedDBName, Creator, ItemPedDBType);
    if Result then
      Result := PSDatabase.Open(DBItp, ItemPedDBName, dmModeReadWrite);
  end;

  if not Result then
  begin
    ShowSystemError(PSDatabase.LastError);
    Exit;
  end;

  PSDatabase.SetFieldDefs(DBItp, FieldDefs[0],
    SizeOf(FieldDefs) div SizeOf(FieldDefs[0]));
end;

function Close: Boolean;
begin
  Result := PSDatabase.Close(DBItp);
end;

function ProcuraCodigo(Codigo: UInt16): Boolean;
begin
  PSDatabase.First(DBItp);
  while not PSDatabase.EOF(DBItp) do
  begin
    if PSDatabase.FieldUInt16(DBItp, Itp_NumeroPed) = Codigo then
    begin
      Result := True;
      exit;
    end;
    PSDatabase.Next(DBItp);
  end;
  Result := False;
end;
end.
```

Listagem 7. Código de todos os botões da tela de Pedidos

```
procedure BtNovoSelect;
begin
  PsDataBase.Insert(DBPed);
end;
procedure BtPedItensSelect;
begin
 FrmGotoForm(FrmPedidoItem);
end;
procedure BtInicioSelect;
begin
 PsDataBase.First(DBPed);
end;
procedure BtAnteriorSelect;
begin
 PsDataBase.Prior(DBPed);
end;
procedure BtProximoSelect;
begin
 PsDataBase.Next(DBPed);
end;
procedure BtFimSelect;
begin
 PsDataBase.Last(DBPed);
end;
procedure BtGravarSelect;
begin
 PsDataBase.Post(DBPed);
end;
procedure BtExcluirSelect;
begin
 PsDataBase.Delete(DBPed);
end;
procedure BtnVoltarSelect;
begin
 FrmGotoForm(FrmPrincipal);
end;
```

Listagem 8. Código dos botões Produtos e Pedidos

```
..
uses
  UDadosClientes, UDadosProdutos,
  UDadodsPedidos;
..
procedure btnProdutosSelect;
begin
  FrmGotoForm(FrmDadosProdutos);
end;

procedure btnPedidosSelect;
begin
  FrmGotoForm(FrmDadosPedidos);
end;
```

de artigos trabalharemos muito com a *List* e entenderemos suas principais funcionalidades.

Após entendermos o funcionamento dos novos componentes, vamos codificar nosso fonte. Na **Listagem 7** encontramos o código de cada botão incluso no exemplo. Codifique conforme necessário.

Precisamos agora preparar o sistema para que possa abrir as novas telas a partir do formulário principal. Portanto, retorne o formulário principal e codifique, no evento *OnSelect* dos botões Produtos e Pedidos, uma chamada ao método *FrmGotoForm* indicando qual formulário será aberto. Não esqueça que deverá adicionar ao *Uses* do formulário principal as duas *Units* para referência. Se preferir veja o trecho código dos dois botões na **Listagem 8**.

Apenas para lembrar, devemos acrescentar o formulário de pedidos no Unit principal do projeto, na função ApplicationHandleEvent, para que o sistema operacional do PalmOS reconheça o formulário de Pedidos e Produtos como parte da aplicação. Abra a unit Vendas e adicione a chamada ao case..of no método. Veja o código completo na **Listagem 9**.

Aproveitando que estamos mexendo na *Unit Vendas*, precisamos fazer dois últimos ajustes. Precisamos abrir e fechar as tabelas *ProdutosDB*, *PedidosDB* e *ItemPedDB* nos métodos *StopApplication* e *StartApplication*.

Localize-os e altere conforme a **Listagem 10**. Observe que o evento *StopApplication* fecha as tabelas usando o método *Close* e o evento *StartApplication* abre-as usando o método *Open*. A única particularidade, é que na abertura das tabelas, nós testamos se a tabela pode ser aberta. Em caso negativo enviamos uma mensagem informando que a tabela não pode ser encontrada.

Listagem 9. Código do método ApplicationHandleEvent alterado

```
function ApplicationHandleEvent(var Event: EventType):
  Boolean;
var
  FormID: UInt16;
  Form: FormPtr;
begin
  Result := False;
  if Event.eType = frmLoadEvent then
  begin
    FormID := Event.frmLoad.formID;
    Form := FrmInitForm(FormID);
    FrmSetActiveForm(Form);
    case FormID of
      FrmPrincipal: FrmSetEventHandler(Form,
        uPrincipal.HandleEvent);
      FrmClientes: FrmSetEventHandler(Form,
        UDadosClientes.HandleEvent);
      FrmDadosProdutos: FrmSetEventHandler(Form,
        UDadosProdutos.HandleEvent);
      FrmDadosPedidos: FrmSetEventHandler(Form,
        UDadodsPedidos.HandleEvent);
    end;
    Result := True;
  end.
end;
```

Listagem 10. Código dos eventos StopApplication e StartApplication alterados

```
function StartApplication: Boolean;
begin
  { Abre o banco de dados de Clientes. Se não conseguir
    abrir, }
  { não executa a aplicação}
  if not ClientesDB.Open then
  begin
    ShowMessage('Tabela de Clientes não encontrada!');
    Result := False;
    Exit;
  end;
  if not ProdutosDB.Open then
  begin
    ShowMessage('Tabela de Produtos não encontrada!');
    Result := False;
    Exit;
  end;
  if not PedidosDB.Open then
  begin
    ShowMessage('Tabela de Itens não encontrada!');
    Result := False;
    Exit;
  end;
  if not ItemPedDB.Open then
  begin
    ShowMessage('Tabela de Itens não encontrada!');
    Result := False;
    Exit;
  end;
  Result := PSApplication.CheckROMVersion($2003000);
  if not Result then
  begin
    FrmCustomAlert(AlertIncompatible,'2.0', nil, nil);
    Exit;
  end;

  FrmGotoForm(FrmPrincipal);
end;

procedure StopApplication;
begin
  ClientesDB.Close;
  ProdutosDB.Close;
  PedidosDB.Close;
  ItemPedDB.Close;
  FrmCloseAllForms;
end;
```

Conclusão

Nesse artigo criamos a tela de consulta a produtos, o banco de dados de pedidos e itens do pedido, conhecemos os conceitos de formulários do Palm, destrutivo e não destrutivo, e criamos a tela do cabeçalho do pedido. No próximo artigo criaremos a tela de Itens do Pedido e construiremos o *Conduit*. Até o próximo artigo, e bons códigos a todos! ●

Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback

Existem coisas que não conseguimos ficar sem!

...só pra lembrar, sua assinatura pode estar acabando!

Renove Já!

www.devmedia.com.br/renovacao

DevMedia GROUP

Para mais informações:
www.devmedia.com.br/central

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Envio de E-mails com componentes da paleta Indy

Veja como enviar e-mails utilizando os componentes da paleta Indy

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da Equipe Editorial ClubeDelphi.

O envio de e-mails automatizado em sistemas é simplesmente uma ferramenta fantástica. Através de componentes e métodos, o desenvolvedor disponibiliza em seu sistema uma agenda de e-mails, que ao ser configurada dispara e-mails com relatórios, avisos, entre outros, automaticamente aos seus destinatários. No Delphi, podemos contar com os componentes da paleta *Indy* para criar um sistema de envio de e-mails.

Neste artigo veremos a criação de um sistema simples de envio de mensagens eletrônicas (e-mail), onde faremos além das configurações normais de remetente, destinatário, assunto e mensagem a configuração de autenticação dos usuários por SMTP, também a configuração para que o e-mail possa ser enviado com anexos.

Criando a aplicação

Utilizaremos o Delphi 7 para a criação do sistema de envio de e-mail com componentes da paleta *Indy*. Com o Delphi aberto, utilize o menu *File|New>Application* para criar uma nova aplicação. Altere a propriedade *Name* do formulário principal para "frmSendMail" e o *Caption* altere para "Envio de E-mails". Salve a *Unit* do formulário principal como "uEmail.pas" e o projeto salve como "prjEmail.dpr".

Nota do DevMan

De forma bem simples, o envio de e-mails funciona basicamente através de um cliente de e-mail acessado pelo browser de internet ou através de gerenciadores instalados e configurados nas estações de trabalho, tais como Microsoft Outlook, Outlook Express, Mozilla Thunderbird entre outros. O usuário que necessita de sua utilidade precisa ter uma credencial de cadastro (usuário e senha) em um servidor de e-mails SMTP ("Simple Mail Transfer Protocol") para o envio dos e-mails. Toda vez que a ação de envio for requisitada ao gerenciador, uma autenticação será realizada no servidor, e este por sua vez irá proceder com o envio do texto/arquivo para o destinatário pré-informado.

Ajuste as dimensões do formulário alterando sua propriedade *Height* para *580* e *Width* para *510*. Logo no cabeçalho do formulário arraste da paleta *Standard* um componente *GroupBox*("gbSMTP"), alterado sua propriedade *Caption* para "Autenticação de Usuários SMTP". Dentro do componente *gbSMTP* adicione ao topo um componente *CheckBox*("CkSMTP"), alterando sua propriedade *Caption* para "Meu servidor requerer uma autenticação SMTP para envio de e-mails", que será utilizado nos casos em que é necessária autenticação para realizar o envio da mensagem. Adicione logo abaixo do *ckSMTP* três componentes *Label*, alterando a propriedade *Caption* de ambos para "Host", "Usuário" e "Senha", respectivamente. Logo abaixo cada *Label* adicionado iremos inserir três componentes de texto. Arraste da paleta *Additional* um componente *MaskEdit*("edtHost") e altere sua propriedade *EditMask* para "999\.999\.999\.999;1;_", sendo esta a máscara para formatação do endereço *TCP/IP* do servidor de autenticação SMTP a ser utilizado para o envio. Altere também a propriedade *Text* do componente para "000.000.000.000". Arraste agora da paleta *Standard* dois componentes *Edit*, alterando a propriedade *Name* de ambos para "edtUsuario" e "edtSenha", respectivamente de acordo com os *Captions* dos *Label's* acima. Seu *gbSMTP* ficará semelhante à imagem ilustrada na **Figura 1.**

Adicione imediatamente abaixo ao *gbSMTP* um novo componente *GroupBox*("gbEmail"), alterando seu *Caption* para "Cabeçalho do E-mail". Dentro do *gbEmail* adicione ao topo um *Label* com o *Caption* "De:" e imediatamente abaixo arraste da paleta *Standard* um *Edit*("emailDe") e remova o valor da sua propriedade *Text* deixando-o em branco. Logo abaixo ao *emailDe*, arraste um novo *Label* com o *Caption* "Para:" inserindo em seguida da paleta *Standard* um componente *Memo*("edtPara") e removendo seu texto na propriedade *Lines*. Altere também a propriedade *ScrollBars* do *edtPara* para "*ssVertical*" fazendo com que uma barra de rolagem no sentido vertical seja exibida no componente. Ainda abaixo do mesmo, arraste um *Label* com o *Cap-*

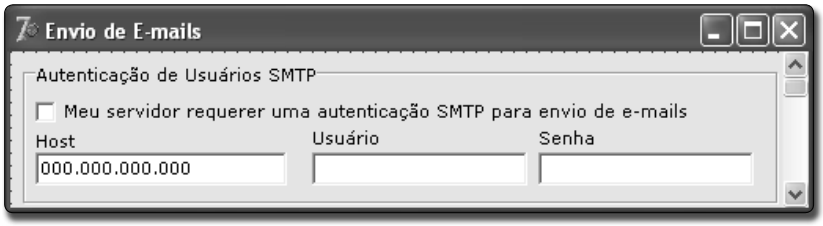


Figura 1. Componentes para autenticação SMTP

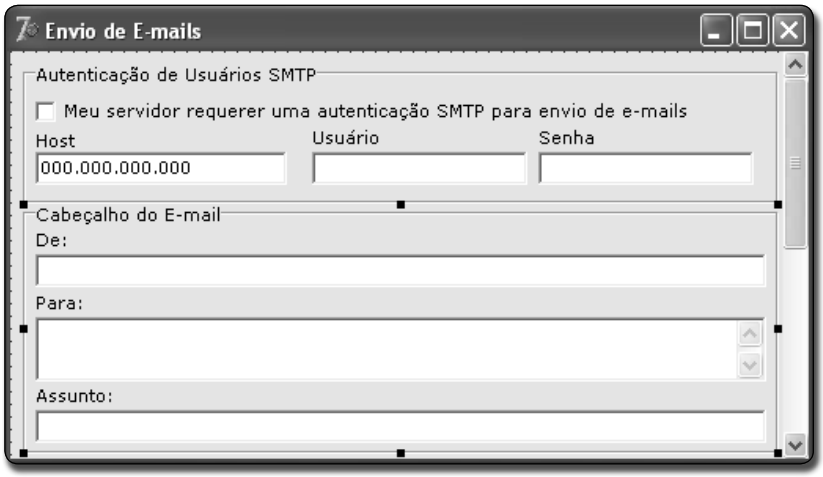


Figura 2. Cabeçalho do E-mail

Figura 3. Formulário de envio de e-mails

tion "Assunto" e insira um componente *Edit*("edtAssunto"), removendo também o seu *Text*. Ao final desta configuração, seu formulário deverá estar de acordo com a **Figura 2.**

Para inclusão de anexos ao e-mail, adicione logo abaixo ao *gbEmail* um novo *GroupBox*("gbAnexo"), alterando seu *Caption* para "Anexo" e arraste para dentro do mesmo um componente *ListView*("emailAnexo") da paleta *Win32*. Ainda abaixo ao *gbAnexo*, adicione mais um *GroupBox*("gbMensagem") e defina seu *Caption* para "Mensagem". Arraste para dentro do *gbMensagem* um componente *RichEdit* ("emailMensagem") da paleta *Win32* e remova o texto da sua propriedade *Lines*.

Ao final do formulário, adicione ainda da paleta *Additional* dois componentes *BitBtn*("btnEnviar" e "btnCancelar"). Altere a propriedade *Caption* de ambos para "ENVIAR" e "CANCELAR", respectivamente. Inclua, através da propriedade *Glyph*, uma imagem a cada um dos botões. Ao final seu formulário estará com o *Layout* e semelhante a **Figura 3.** Veremos a seguir os componentes necessários para envio do e-mail e adição dos anexos à mensagem.

Componentes para envio do e-mail e adição de anexos

Veremos a seguir os componentes necessários a serem adicionados no formulário para o envio do e-mail. Ao lado de cada componente, veja para qual função será utilizado e quais as configurações necessárias:

- *IdSMTP* ("CompSMTP") da paleta *IndyClients:* será o responsável pela comunicação direta com o servidor de autenticação e envio dos e-mails SMTP. As configurações de suas propriedades serão realizadas em tempo de execução e veremos a seguir.

- *IdMessage* ("CompMensagem") da paleta *Indy Misc:* componente no qual criamos o e-mail. Nele que serão adicionados o assunto, mensagem, entre outras partes do e-mail. Sua configuração também será realizada em tempo de execução.

- *IdAntiFreeze* ("AntiGelo") da paleta *Indy Misc:* sua utilidade é não deixar a aplicação "congelar" enquanto o e-mail está sendo enviado. O componente mantém o formulário atualizado, permitindo que até durante o envio o usuário possa continuar interagindo com o sistema.

- *OpenDialog* ("OpenAnexo") da paleta *Dialogs:* será utilizado para localizar os anexos a serem adicionados ao e-mail. Não requer nenhuma configuração especial, pois será utilizado apenas para localizar os arquivos.

- *PopupMenu* ("MenuAnexo") da paleta *Standard:* relacione o componente à propriedade *PopupMenu* do componente *emailAnexo*. Será utilizado no clique do botão direito sobre o componente *emailAnexo* para adicionar ou remover arquivos. Com um duplo clique sobre o componente *MenuAnexo*, crie dois itens de menu ("Anexar arquivo" e "Excluir arquivo").

- *ImageList* ("Imagens") da paleta *Win32:* componente onde iremos adicionar duas imagens que serão listadas junto com o componente *MenuAnexo*. Com um duplo clique sobre o componente *Imagens* utilize o botão *Add* e localize dois ícones relacionados aos menus criados no *PopupMenu*. Por *default* você irá encontrar ícones para serem adicionados no caminho *C:\Arquivos de programas\Arquivos comuns\Borland Shared\Images\Buttons*.

Após adicionadas duas imagens, volte ao componente *MenuAnexo* e relacione a propriedade *Images* ao componente *Imagens*. Com um duplo clique sobre o componente, selecione cada um dos

Listagem 1. Adicionando anexos ao e-mail

```
procedure TfrmSendMail.Anexararquivo1Click(Sender: TObject);

var
  files: TListItem;

begin
  if OpenAnexo.Execute then

  begin
    files := emailAnexo.Items.Add;
    files.Caption := OpenAnexo.FileName;
  end;
end;
```

Listagem 2. Código para exclusão de anexo

```
procedure TfrmEmails.Excluir1Click(Sender: TObject);
var
  Arquivo : string;
begin
  Arquivo := ExtractFileName(emailAnexo.Items.Item[
    emailAnexo.ItemIndex].Caption);
  if Application.MessageBox(PAnsiChar('Deseja excluir
    o arquivo ' + Arquivo + '?'), 'Excluir anexo',
    MB_YESNO) = idYes then
    emailAnexo.Items.Delete(emailAnexo.ItemIndex);
end;
```

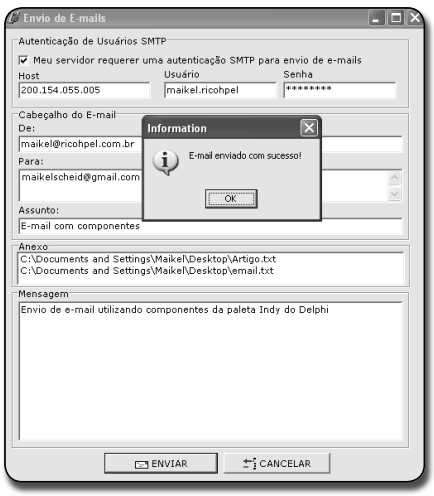


Figura 4. Envio do e-mail

itens de menu adicionados e na propriedade *ImageIndex* selecione o ícone correspondente.

Anexando arquivos ao E-mail

Com um clique duplo sobre o componente *MenuAnexo*, selecione o item de menu "Anexar arquivo" e adicione a seu evento *OnClick* o código da **Listagem 1**, obedecendo a criação de uma variável do tipo *TListItem* para que seja instanciado o arquivo selecionado e exibido no *emailAnexo*.

O código é bem simples, apenas criamos uma variável do tipo *TListItem* e chamamos o método *Execute* do controle *OpenAnexo*. Após a seleção de um arquivo na caixa de diálogo, adicionamos um novo item ao *emailAnexo*, componente *ListView*, e configuramos o seu *Caption* com o nome do arquivo selecionado.

Já no item de menu "Excluir arquivo" removemos o arquivo da lista de anexos. Primeiro copiamos o nome do arquivo selecionado em uma variável do tipo *String* e em seguida chamamos o método *Delete* do componente *emailAnexo*. Veja a **Listagem 2**.

Enviando o e-mail

Para realizar o envio do e-mail precisamos criar em tempo de execução algumas configurações relacionadas aos componentes adicionados. Adicione ao evento OnClick do btnEnviar o código da **Listagem 3**, que encontra-se comentado de acordo com cada ação que está ocorrendo durante o processo de envio do e-mail.

É muito importante que todo o código seja mantido em um bloco *try..finally* sendo que entre o *finally..end* deverá estar a linha para desconectar ao servidor SMTP. Tanto num caso de sucesso do envio do e-mail ou num caso em que ocorra uma falha, o componente é desconectado não deixando nenhuma porta de ligação aberta.

O cancelamento do envio do e-mail na verdade trata-se de uma saída do formulário. Para tanto adicione ao evento *OnClick* do *btnCancelar* uma chamada ao método *Close*.

Configurados todos os eventos da aplicação, execute a mesma e faça o teste de envio. Certifique-se antes de que está usando um *IP* válido para autenticação SMTP como servidor de saída das mensagens. Estando corretas todas as informações, seu e-mail será enviado com sucesso (**Figura 4**).

Conclusão

Neste artigo aprendemos a criar uma aplicação de envio de e-mails utilizando os componentes da paleta *Indy* do Delphi. Com estes códigos você poderá implementar novas funcionalidades, como por exemplo opções de solicitação de entrega e leitura da mensagem, além da formatação do texto a ser enviado. Aplique este recurso ao seu sistema deixando-o mais funcional e atrativo para seu cliente. Abraço e até a próxima. ●

Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback

Listagem 3. Enviando o e-mail

```
procedure TfrmSendMail.btnEnviarClick(Sender: TObject);
var
  i : integer;
begin
  try
    { Verifica se o campo remetente foi preenchido }
    if emailDe.Text <> '' then
    begin
      { Verifica se o campo destinatásio foi preenchido }
      if emailPara.Text <> '' then
      begin
        { Verifica se o campo assunto foi preenchido }
        if emailAssunto.Text <> '' then
        begin
          { Verifica se a mensagem foi preenchida }
          if emailMensagem.Lines.Text <> '' then
          begin
            { Verifica se o servidor de SMTP requer autenticação. De acordo com a
              seleção, o tipo de autenticação do componente será alterada }
            if ckSMTP.Checked then
              CompSMTP.AuthenticationType := atLogin else
              CompSMTP.AuthenticationType := atNone;
            { Configura o Host do servidor de SMTP a ser utilizado para o envio do e-mail }
            CompSMTP.Host := edtHost.Text;
            { Configuração do usuário e senha para autenticação no Host de SMTP }
            CompSMTP.Username := edtUsuario.Text;
            CompSMTP.Password := edtSenha.Text;
            { Faz a conexão ao servidor SMTP }
            CompSMTP.Connect;
            { Verifica se o servidor SMTP foi conectado }
            if CompSMTP.Connected then
            begin
              { Configura a mensagem a ser enviada. Passagem de valores as propriedades }
              CompMensagem.Clear;
              CompMensagem.Priority:=mpHighest;
              { Low = baixo / High = alto / mpHighest = urgente }
              CompMensagem.ContentType := 'text/html';
              CompMensagem.From.Text := emailDe.Text;
              CompMensagem.Recipients.EMailAddresses := emailPara.Lines.Text;
              CompMensagem.Subject := emailAssunto.Text;
              CompMensagem.Body.Text := emailMensagem.Text;
              { Anexa os arquivos ao e-mail }
              for i := 0 to emailAnexo.Items.Count - 1 do
                TIdAttachment.Create(CompMensagem.
                MessageParts, TFileName(emailAnexo.
                Items[i].Caption));
              { Faz o envio do e-mail }
              CompSMTP.Send(CompMensagem);
              MessageDlg('E-mail enviado com sucesso!',mtInformation, [mbOK], 0);
            end;
          end
          else
            ShowMessage('Informe a mensagem do e-mail');
        end
        else
          ShowMessage('Informe o assunto do e-mail!');
      end
      else
        ShowMessage('Informe o destinatário do e-mail!');
    end
    else
      ShowMessage('Informe o e-mail do remetente!');
  Finally
    { desconecta do servidor SMTP }
    CompSMTP.Disconnect;
  end;
end;
```

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Editor de Textos com RichEdit

Veja como criar seu próprio editor de textos com o componente RichEdit

Muitas vezes utilizamos em nossas aplicações o componente *RichEdit*, que possibilita aos programadores aplicá-los em inúmeras situações durante o desenvolvimento de suas aplicações. Entre as diversas utilidades estão formulários de cadastros como notícias, formulários de envio de e-mail ou a criação de seu próprio editor de textos. Com o componente *RichEdit* você pode criar e aperfeiçoar muitas das utilidades que o *Notepad* ("Bloco de Notas") do Windows possui. Pode inovar na criação de um *Layout* e na aplicação de ferramentas e configurações na digitação de textos, salvá-los e reabri-los sem perder nenhuma formatação.

Veja neste artigo a criação de um simples editor usando o componente *RichEdit* e aprenda maneiras de como utilizar suas propriedades codificando situações de formatação de textos, tais como alteração da fonte, tamanho e cor da letra, utilização de estilos de formatação como negrito, sublinhado entre outros. Também alinhamentos e espaçamentos de parágrafos. Faremos uso também, em algumas situações, de alguns componentes da paleta *Dialogs* do Delphi com a função de definir o *Layout* de impressão, salvar e abrir documentos salvos pelo editor de textos ou até mesmo por outros editores.

Criando a aplicação

A criação do editor de textos trata-se de uma simples aplicação na qual utilizaremos o Delphi 7 para a criação do aplicativo Win32. Não será necessária a utilização de nenhuma espécie de banco de dados ou instalação de componentes de terceiros. Serão apenas utilizados componentes nativos das paletas do Delphi. Crie a aplicação no menu *File|New>Application* e altere a propriedade *Caption* do formulário principal para "Meu editor de Textos" e em seguida nomeie o formulário para "frmEditor". Salve a *Unit* do formulário como "uPrincipal.pas" e o

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da Equipe Editorial ClubeDelphi.

projeto como "TextEditor.dpr". Adicione ao formulário principal um componente *MainMenu*("MainMenu1") da paleta *Standard* e com duplo clique do mouse crie uma estrutura de menus conforme ilustrado na **Figura 1.** Esta estrutura de menus também será utilizada mais adiante para a criação dos atalhos de acesso rápido.

Definida agora toda a estrutura de menus e opções do nosso editor de textos, vamos adicionar da paleta *Win32* um componente *ToolBar*("BarraButtons") para que possamos criar alguns atalhos de acesso rápido a algum menus do sistema. Com o clique do botão direito sobre a *ToolBar* selecione a opção *New Button*, criando assim um novo botão ("btNovo") na barra de atalhos. Crie logo após mais três botões utilizando o mesmo método. Nomeie os botões como "btAbrir", "btSalvar" e "btImprimir", respectivamente. Dessa forma acabamos de criar os botões de atalho para os principais itens do menu "Arquivo" e vamos criar agora para os itens do menu *Editar*.

Antes de criar um novo botão na barra de atalhos, vamos adicionar um separador a fim de organizar melhor e categorizar por grupos de funções na barra de atalhos. Clicando sob a barra com o botão direito do mouse e selecionando a opção *New Separator*.

Criado um separador logo após o primeiro grupo de botões, criaremos agora os botões para relacionar atalhos aos itens do menu *Editar*. Crie três botões destinados a este grupo com os nomes "btCopiar", "btRecortar" e "btColar". Logo após adicione um novo separador ao menu.

Sendo nossa idéia a de criar um editor de textos, é interessante que para uma fácil edição do mesmo o usuário tenha sempre de fácil acesso os tipos de fontes para que a qualquer momento possa alterá-la em trechos do seu texto. Para oferecer esta opção ao usuário, adicionaremos também à barra de atalhos um componente *ComboBox*("ckFontes") da paleta *Standard* que será povoado logo mais com toda a lista de fontes instaladas no Windows. Adicionado o componente, crie um novo separador na barra de atalhos. Além da alteração do tipo de fonte em um texto, o usuário precisa também ter sempre de fácil acesso a alteração do tamanho da fonte que está utilizando, e para isso adicione à barra um novo componente *ComboBox(*"ckTamanho"), e na sua propriedade *Items* adicione as variações de tamanhos que deseja disponibilizar. No editor de itens do componente adicione, separados por linha, os seguintes valores:

```
6, 8, 10, 11, 12, 14, 16, 18, 20, 22, 24,
28, 30, 32, 40.
```

A alteração das cores de um texto que se está digitando também é uma característica muito importante durante a criação de um editor. Inclua na barra de atalhos um componente *ColorBox*("ckCor") da paleta *Additional* e em seguida insira mais um separador à barra.

Para permitir a formatação e aplicação de estilos de formatação para textos com negrito, itálico, sublinhado e riscado, adicione à barra de atalhos quatro botões, "btNegrito", "btItalico", "btSublinhado" e "btRiscado". Insira um novo separador. Por último, inclua na barra de atalhos mais três botões "btEsquerda", "btCentro" e "btDireita" que serão utilizados com a finalidade de alinhar o texto para esquerda, centralizar e organizar à direita.

Nossa aplicação está aparecendo agora de uma forma um tanto estranha, temos apenas o menu e uma barra de atalhos com vários botões, sendo que nenhum deles ainda dispõe de qualquer tipo de identificação visual para sua funcionalidade. Para definir um estilo de *layout* e caracterizar cada botão de atalho, arraste para o formulário um componente *ImageList*("AtalhoImages") da paleta *Win32*. Com um clique duplo sobre o mesmo abra seu editor e adicione uma série de imagens de acordo com as funcionalidades que queremos atribuir ao nosso editor de textos. As imagens a serem adicionadas deverão estar com formatos de arquivo como *.bmp ou *.ico e poderão ser facilmente localizadas no diretório de imagens instaladas pelo próprio Delphi, encontradas no caminho padrão em *C:\Arquivos de programas\Arquivos comuns\Borland Shared\Images\Buttons*. Cada imagem adicionada para o componente *ImagemList* recebe um número de identificação que usaremos para definir os atalhos nos botões (**Figura 2**). Selecionando agora o componente "BarraButtons" relacione a propriedade *Images* do mesmo ao *AtalhoImages*, observando que alguns atalhos já assumiram de forma automática alguns ícones de identificação.

Para editar os ícones de cada botão e adequá-los ao ícone de sua respectiva funcionalidade, clique sobre a propriedade *ImageIndex* e na lista de imagens disponíveis selecione qual corresponde à sua função. Ao final da edição das imagens, a barra de atalhos do sistema deverá ficar estruturada de forma semelhante a **Figura 3**.

A configuração dos ícones de menu também deverá ser realizada no componente *MainMenu1*, onde também deverá relacionar sua propriedade *Images* ao "AtalhoImages" e com um duplo clique ao abrir o editor de menus, selecione cada um deles alterando o ícone relacionado a sua propriedade *ImageIndex*. Alterados e configurados todos os ícones de identificação de menus e atalhos do editor, vamos agora adicionar o principal componente da nossa aplicação ao formulário. Na paleta *Win32* adicione um componente *RichEdit*("TextEditor"), defina sua propriedade *Align* para *alClient*

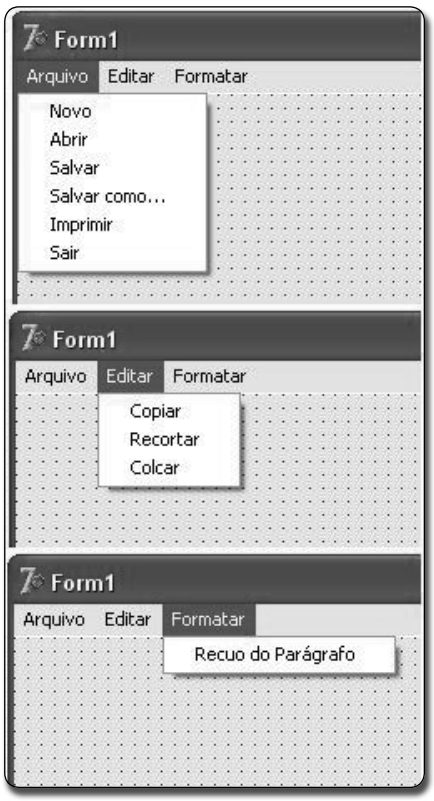


Figura 1. Estrutura de menus do sistema

e remova o texto padrão exibido na sua propriedade *Lines*.

Posicionado agora sob a paleta *Dialogs*, adicione ao formulário os componentes *PrintDialog*("dlgPrint") que será utilizado para configuração e envio do texto à impressora, *SaveDialog*("dlgSave") utilizado para salvar e armazenar o texto do editor como um documento em qualquer unidade de armazenamento do computador e também adicione um *OpenDialog*("dlgOpen") utilizado para localizar documentos armazenados no computador e reabri-los a fim de ler ou continuar com a edição do texto.

O formato de arquivo que utilizaremos será o .rtf, ao qual iremos definir um filtro para a propriedade *Filter* dos componentes *dlgSave* e *dlgOpen*. Abrindo o editor de filtros dos componentes, defina dois filtros em linhas diferentes, obedecendo o *Filter Name* como "Arquivos de Texto (*.rtf)" e "Todos os arquivos" e na coluna *Filter* obedeça *.rft e *.*, respectivamente (**Figura 4**). Essas configurações irão fazer com que apenas arquivos com esta extensão sejam exibidos na hora de salvar ou localizar documentos nas unidades de armazenamento do computador. Configure ainda em ambos os componentes a propriedade *DefaultExt*, atribuindo-lhes o valor *.rtf que será a extensão padrão que adotamos para nosso sistema.

Ao final da adição e configuração de todos os componentes ao formulário, sua aplicação deverá estar semelhante à aplicação da **Figura 5**, na qual passaremos a partir deste momento a codificar as funções do nosso editor.

Codificando a aplicação

A codificação de todo o processo do editor de textos não será muito complicada, vemos que se trata de pequenos blocos de comandos que serão facilmente entendidos e poderão ser aplicadas a várias outras situações que você poderá se deparar enquanto desenvolver sua aplicação. A primeira codificação que veremos será quanto a criação do formulário (evento *OnCreate*) onde iremos realizar algumas configurações que serão os *Default* do nosso editor. Acessando o editor de códigos do formulário, declare na seção *private* a seguinte *procedure* que será responsável por capturar todas as fontes instaladas no Windows e preencher a lista no componente *ckFontes*.

```
procedure ObtemFontes;
```

Posicionado sobre a declaração da *procedure* utilize as teclas *Ctrl+Shift+C* para que o Delphi crie o cabeçalho do procedimento, e adicione ao mesmo o código da **Listagem 1**, onde criaremos uma *function* derivada dentro da *procedure* que será referenciada como forma de ponteiro e referência a um espaço de memória a ser locado no sistema. Ainda dentro da *procedure* criamos uma variável do tipo *LongWord* que é usada para carregar os nomes das fontes e logo mais são atribuídas ao componente *ckFontes*.

No evento *OnCreate* do formulário, adicione as seguintes linhas de código, que irão chamar a *procedure* do carregamento das fontes recém criada e definir nos componentes *ckFontes* e *ckTamanho* o tipo e tamanho das configurações padrão que estão definidas no sistema, trazendo-as selecionadas:

```
ObtemFontes;
ckFontes.Text  := DefFontData.Name;
ckTamanho.Text :=
IntToStr(-MulDiv(DefFontData.Height, 72,
  Screen.PixelsPerInch));
```

Nas próximas linhas de código, vamos atribuir funcionalidades aos menus do sistema (*MainMenu1*), e para os casos de existir um atalho rápido na barra correspondente a um item do menu original (menu *Abrir* por exemplo), iremos apenas relacionar o botão ao item através da sua propriedade *MenuItem* que veremos também mais a seguir.

Clique duas vezes no *MainMenu1* e com ele aberto navegue ao item *Arquivo>Novo*. Automaticamente o Delphi nos cria o cabeçalho do evento *OnClick* desse item ao qual digitaremos as seguintes linhas de comando:

```
if TextEditor.Modified then
  ShowMessage('Deseja salvar arquivo')
else
  TextEditor.Lines.Clear;
```

No código verificamos se houve alguma alteração do texto digitado e perguntamos se o usuário deseja salvar o conteúdo ou senão limpamos todos os valores existentes na linhas.

Repita esse processo e digite o código da **Listagem 2** no evento *OnClick* do menu *Arquivo>Abrir*. Esse evento irá executar uma caixa de pesquisa do componente *dlgOpen* e ao selecionar um arquivo que o usuário deseja abrir, verificamos se a extensão do mesmo corresponde ao tipo de arquivos com que estamos trabalhando carregando-o no editor, caso contrario você será notificado de que o formato do arquivo selecionando não corresponde aos ar-

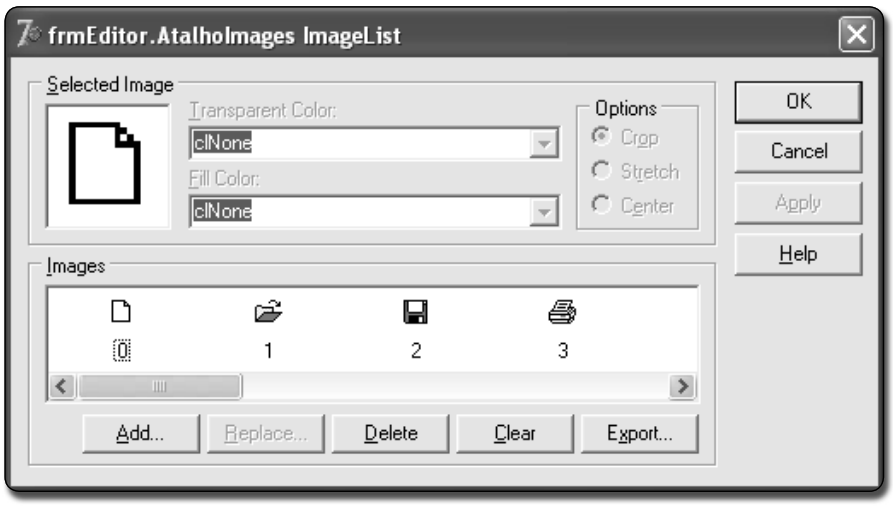


Figura 2. Adicionando imagens para ImageList

Figura 3. Barra de atalhos do Editor de Textos

quivos suportados pelo nosso editor e a ação será ignorada.

Para salvar o conteúdo digitado no editor, atribua ao evento *OnClick* do menu *Arquivo>Salvar* o código da **Listagem 3** que ao executar o componente de diálogo *dlgSave* irá verificar se o nome do arquivo fornecido pelo usuário já existe na pasta selecionada. Havendo o arquivo ele será sobrescrito. Em caso contrário será criado um novo arquivo com o nome fornecido na pasta de destino selecionada pelo usuário. Por fim a propriedade *Modified* do editor será repassada como *False*, fornecendo assim que não existe conteúdo modificado sem ser salvo caso deseje criar um novo documento.

A codificação do menu *Salvar Como* também se dá de forma bastante simples, pois nenhuma verificação precisa ser realizada, necessitando apenas executar o diálogo do *dlgSave* e salvar o conteúdo na pasta selecionada com o nome fornecido. O código a seguir deverá ser adicionado ao seu evento *OnClick:*

```
if dlgSave.Execute then
begin
  TextEditor.Lines.SaveToFile(
    dlgSave.FileName);
  TextEditor.Modified := false;
end;
```

Pelo fato de termos adicionado ao formulário um componente para diálogo da impressora, a impressão também é de forma muito simples. Basta adicionar ao evento *OnClick* do menu *Imprimir* o código a seguir, que será responsável por chamar o diálogo e enviar a impressora:

```
  if (dlgPrint.Execute) then
    TextEditor.Print(Text);
```

No evento *OnClick* do menu *Sair* adicione o código da **Listagem 4.** Aqui verificamos se houve alterações no componente *RichText*. Em caso positivo perguntamos ao usuário se ele deseja salvar o arquivo, caso contrário apenas fechamos o aplicativo.

Para codificação dos três dos itens do menu *Editar* observe na **Listagem 5** os códigos referentes a cada item de menu digitando os respectivos códigos para as ações de *Copiar, Recortar* e *Colar*.

O componente *RichText* possui três métodos distintos para se tratar mensagens em memória que são *CopyToClipBoard*, *CutToClipBoard* e *PasteToClipBoard* que, respectivamente, copiam, recortam e colam o conteúdo *de* ou *para* a memória ("ClipBoard").

No menu *Recuo* do parágrafo adicione ao evento *OnClick* as seguintes linhas de código, que irão abrir uma caixa de digitação para que o usuário possa definir qual espaçamento deseja que seja dado ao seu texto. Antes de adicionar o código, declare nas *uses* do projeto a *Unit Dialogs.*

```
//Adicionar uses Dialogs
TextEditor.Paragraph.FirstIndent :=
  TextEditor.Paragraph.FirstIndent +
    StrToInt(InputBox('Digite o
    espaçamento','Recuo do Texto (em mm)',
    IntToStr(TextEditor.Paragraph.
    FirstIndent)));
```

Nota do DevMan

Se você estiver trabalhando no Windows VISTA, será necessário executar o Visual Web Developer como administrador, para que a aplicação tenha direitos suficientes para gravar no log de Eventos.

Para executar o Visual Web Developer como administrador, clique com o botão direito sobre o seu ícone no menu, e escolha a opção "Executar como administrador" ou "Run as administrator".

Figura 4. Configuração dos filtros

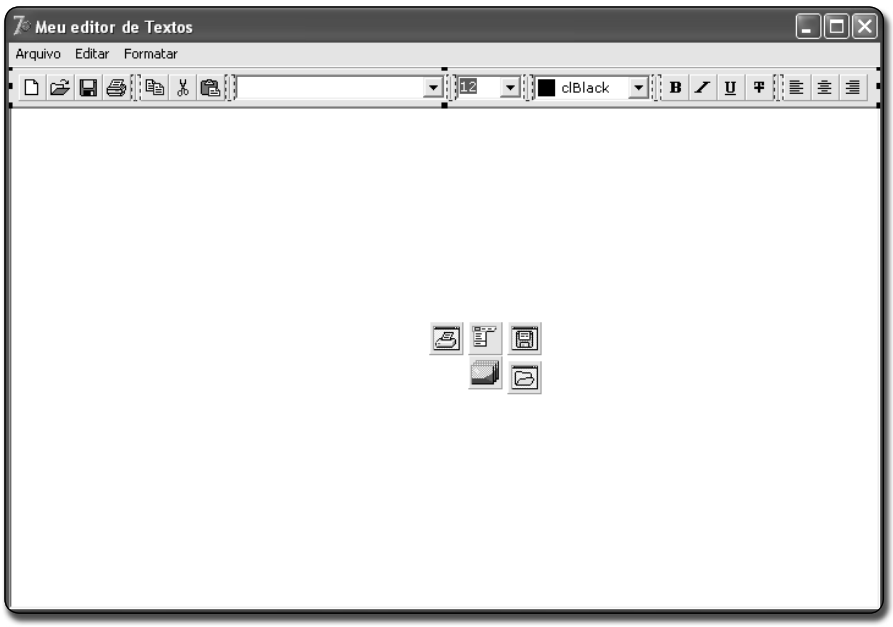


Figura 5. Estrutura do Editor de Textos

Listagem 1. Implementação da procedure ObtemFontes

```
procedure TfrmEditor.ObtemFontes;
  function EnumFontsProc(var LogFont: TLogFont; var
    TextMetric: TTextMetric; FontType: Integer;
    Data: Pointer): Integer; stdcall;
  begin
    TStrings(Data).Add(LogFont.lfFaceName);
    Result := 1;
  end;
var
  DC: HDC;
begin
  DC := GetDC(0);
  EnumFonts(DC, nil, @EnumFontsProc,
    Pointer(ckFontes.Items));
  ReleaseDC(0, DC);
  ckFontes.Sorted := True;
end;
```

No código anterior atribuímos o valor digitado pelo usuário à propriedade *FirstIdent* do *Paragraph*. O *Paragraph* é uma propriedade do controle *RichText* responsável por alterar as configurações de parágrafo, como o próprio nome sugere.

Finalizada a atribuição dos códigos aos itens de menu do componente *MainMenu1*, precisamos relacionar os atalhos rápidos àqueles itens que já foram codificados, como é o caso dos atalhos de *Novo*, *Abrir* e *Salvar*.

Selecionando o botão correspondente, vá até a propriedade *MenuItem* e localize o nome do item de menu correspondente. Repita o mesmo procedimento em todos os botões.

Alterando tipo, tamanho e cor da fonte

Temos adicionado a nossa barra de atalhos rápidos três funcionalidades, as quais ainda não foram mencionadas em nenhuma codificação e que são indispensáveis a um editor de textos. Alteração do tipo, tamanho e cor de fontes. Podemos selecionar trechos do texto e facilmente editar qualquer uma das propriedades apenas mudando o item selecionando no *ComboBox* relacionado a opção. Os códigos são simples e aplicados ao evento *OnChange* de cada um dos componentes, conforme poderá observar nos códigos e comentários relacionados na **Listagem 6** a seguir. Os códigos estão comentados e não há segredo.

Também atalhos que ainda não foram codificados, temos os botões para aplicação do estilos de *Negrito, Itálico, Sublinhado* e *Riscado* sobre os trechos de textos selecionado. Na codificação de todos estes botões, verificamos se o estilo já se encontra aplicado aos trechos selecionados, e no caso de já existir, o mesmo será removido deixando o texto sem o estilo clicado. Adicione a cada um dos botões a linha de código correspondente que está prevista na **Listagem 7.** Em resumo, o que estamos fazendo é adicionar ou remover o estilo empregado à propriedade *SelAttributes* do componente *RichText*.

Temos ainda a codificação dos três últimos botões de acesso rápido do nosso sistema, correspondentes ao alinhamento do texto ao lado esquerdo, centralizado e à direita do editor. Para alinhar o texto selecionado, modificamos a propriedade Alignment atribuindo os valores taLeftJustify("esquerda"), taCemter("centro") ou taRigthJustify ("direita") ao evento OnClick dos botões btnEsquerda, btnCentro e btnDireita, respectivamente. Veja o código do botão btnEsquerda. Repita o código para os demais botões trocando apenas a atribuição final:

```
TextEditor.Paragraph.Alignment :=
  taLeftJustify;
```

Após todo esse processo de codificação dos menus e botões de acesso rápido do sistema, acabamos com a criação do nosso próprio editor de textos, que poderá ser testado e utilizado para aplicação de diversos estilos.

Execute-o e faça a digitação de valores,

Listagem 2. OnClick para o menu Abrir

```
procedure TfrmEditor.Abrir1Click(Sender: TObject);
begin
  if dlgOpen.Execute then
  begin
    if ExtractFileExt(dlgOpen.FileName) = '.rtf' then
    begin
      TextEditor.Lines.LoadFromFile(dlgOpen.FileName);
      dlgSave.FileName := dlgOpen.FileName;
      TextEditor.Modified := false;
    end
    else
      MessageDlg('Formato de arquivo não suportado',mtInformation,[mbOk],0)
  end;
end;
```

Listagem 3. OnClick do menu Arquivo|Salvar

```
procedure TfrmEditor.Salvar1Click(Sender: TObject);
begin
  if FileExists(dlgSave.FileName) then
    TextEditor.Lines.SaveToFile(dlgSave.FileName)
  else
  begin
    if dlgSave.Execute then
    begin
      TextEditor.Lines.SaveToFile(dlgSave.FileName);
    end;
  end;
  TextEditor.Modified := false;
end;
```

Listagem 4. Evento ativado ao sair do editor de textos

```
procedure TfrmEditor.Sair1Click(Sender: Tobject);
begin
  if TextEditor.Modified then
  begin
    if Application.MessageBox('Deseja salvar as
      alterações?', 'Salvar',MB_YESNO) = IdYes then
    begin
      if dlgSave.Execute then
        TextEditor.Lines.SaveToFile(dlgSave.FileName);
      Close;
    end
    else
      Close;
  end
  else
    Close;
end;
```

Listagem 5. Copiar, Recortar e Colar

```
procedure TfrmEditor.Copiar1Click(Sender: TObject);
begin
  TextEditor.CopyToClipboard;
end;

procedure TfrmEditor.Recortar1Click(Sender: TObject);
begin
  TextEditor.CutToClipboard;
end;

procedure TfrmEditor.Colar1Click(Sender: TObject);
begin
  TextEditor.PasteFromClipboard;
end;
```

Listagem 6. Alteração do tipo, tamanho e cor da fonte do texto

```
procedure TfrmEditor.ckFontesChange(Sender: TObject);
begin
  { Altera o tipo da fonte no texto selecionado. }
  TextEditor.SelAttributes.Name :=
    ckFontes.Items[ckFontes.ItemIndex];
end;
procedure TfrmEditor.ckTamanhoChange(Sender: TObject);
begin
  { Altera o tamanho da fonte no texto selecionado. }
  TextEditor.selattributes.Size :=
  StrToInt(ckTamanho.Items[ckTamanho.ItemIndex]);
end;

procedure TfrmEditor.ckCorChange(Sender: TObject);
begin
  { Altera a cor da fonte no texto selecionado. }
  TextEditor.selattributes.Color := ckCor.Selected;
end;
```

Listagem 7. Aplicação de estilos ao texto.

```
procedure TfrmEditor.btNegritoClick(Sender: TObject);
begin
  if (fsBold in TextEditor.selattributes.Style) then
  TextEditor.selattributes.Style := TextEditor.SelAttributes.Style - [fsBold] else
  TextEditor.selattributes.Style := TextEditor.SelAttributes.Style + [fsBold];
end;
procedure TfrmEditor.btItalicoClick(Sender: TObject);
begin
  if (fsItalic in TextEditor.selattributes.Style) then
    TextEditor.selattributes.Style := TextEditor.SelAttributes.Style - [fsItalic]
  else
    TextEditor.selattributes.Style := TextEditor.SelAttributes.Style + [fsItalic];
end;
procedure TfrmEditor.btSublinhadoClick(Sender: TObject);
begin
  if (fsUnderline in TextEditor.selattributes.Style)
  then
    TextEditor.selattributes.Style := TextEditor.SelAttributes.Style - [fsUnderline]
  else
    TextEditor.selattributes.Style := TextEditor.SelAttributes.Style + [fsUnderline]
end;
procedure TfrmEditor.btRiscadoClick(Sender: TObject);
begin
  if (fsStrikeOut in TextEditor.selattributes.Style)
  then
    TextEditor.selattributes.Style := TextEditor.SelAttributes.Style - [fsStrikeOut]
  else
    TextEditor.selattributes.Style := TextEditor.SelAttributes.Style + [fsStrikeOut]
end;
```

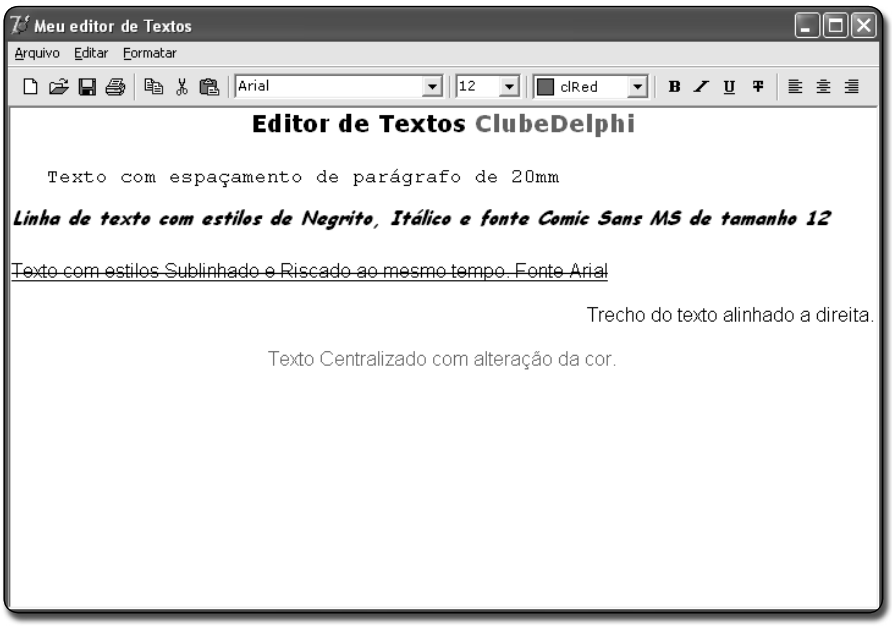


Figura 6. Utilizando funcionalidades do Editor de textos

utilizando todas as propriedades aqui configuradas. Veja que inúmeras ações com o texto podem ser selecionadas (**Figura 6**).

Conclusão

Vimos com essa criação do editor de textos que podemos criar nossas próprias ferramentas de trabalho. Imagine neste caso a substituição do bloco de notas pelo seu próprio editor de textos e quem sabe com mais funcionalidades. Crie agora formulários personalizados e programe novas utilidades ao editor, adicione métodos para marcadores e numeradores de parágrafos, espaçamento do texto entre linhas e letras, pesquisas por palavras, substituição de textos, entre outras inúmeras funcionalidades que poderão ser configuradas.

Abraço e até a próxima. ●

Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Orientação a Objetos no Delphi for PHP

Como aplicar conceitos de POO em aplicações PHP – Parte 2

Rodrigo Carreiro Mourão
(rodrigocarreiro@tdstecnologia.com.br)
Consultor da TDS Tecnologia RJ atuando na área de desenvolvimento de projetos Orientados a Objetos, Design Patterns, MVC. BDS2006 Win32 Product Certified. Borland Instructor Certified. Instrutor de treinamento oficiais CodeGear DelphiWin32, Delphi for PHP, Delphi .Net. Palestrante da Borland Conference 2007.

Como prometido em meu primeiro artigo vamos dar continuidade a este assunto tão extenso que é a orientação a Objetos. Como dito anteriormente toda linguagem dita OO deve estar apoiada nos pilares desta filosofia que são: *Herança, Encapsulamento, Abstração* e *Polimorfismo* e o PHP 5 implementa essas quatro características inclusive o *polimorfismo* que é o foco do nosso artigo.

Muitos programadores vêem a herança como a solução de seus problemas, porém se usada indiscriminadamente ele pode ser um feitiço que se vira contra o feiticeiro, isto porque se não tomarmos os devidos cuidado acabamos por acoplar todo o nosso código através da herança. Porém se há algo que justifique o uso da herança este se chama *polimorfismo*.

O *polimorfismo* é um dos assuntos mais importantes na POO. Com o uso de classes e heranças conseguimos facilmente descrever uma situação da vida real além de conseguir estender projetos por meios da reutilização de código. Mas se de um lado o *polimorfismo* é um dos assuntos mais importantes da POO é também sem dúvida nenhuma um dos mais difíceis de ser compreendido. Isto porque muitos que se aventuram pela primeira vez na OO entram ainda com uma mentalidade *procedural* e enxergando as classes como entidades do banco de dados tornando mais difícil a compreensão da POO como um todo.

Polimorfismo

Teoricamente e a grosso modo entende-se por *polimorfismo "a capacidade que um mesmo método tem de se comportar de maneira diferente dependendo de qual classe ele foi chamado"*. Nós conseguimos isso declarando um método em uma classe base e o sobrescrevemos em uma classe herdada, facilitando assim o desenvolvimento e reaproveitamento códigos.

Antes de mais nada é importante dei-

xar claro que a linguagem que estamos trabalhando é o PHP e que o mesmo é interpretado, fracamente *tipado* e sendo assim não há a necessidade de se declarar variáveis com seus tipos. Mas o que isso tem a ver com *polimorfismo*? Tudo! Observe o código da **Listagem 1**.

No código da **Listagem 1** será levantada uma exceção no evento *OnClick* do botão com a mensagem "Nem todo ser vivo produz som!!!" mesmo um objeto do tipo *THomem* sendo instanciado para a variável *Ser*. Isso acontece porque no Delphi os métodos são estáticos por padrão. Isso significa que independente do objeto que seja instanciado na variável vale para ela o tipo que foi declarado. Para conseguirmos o efeito desejado, ou seja, o método *ProduzirSom* da classe *THomem* seja chamado, é preciso declarar o método *ProduzirSom* na classe *TSerVivo* como *Virtual*.

Isso faz com que o Delphi crie uma VMT("Virtual Method Table") uma tabela com os ponteiros dos métodos que foram declarados como virtuais para que possam ser sobrescritos em tempo de execução. Na **Listagem 2** vemos como ficaria o código.

No PHP nada disso é necessário, como ele e fracamente tipado, ou seja, não

Nota do DevMan

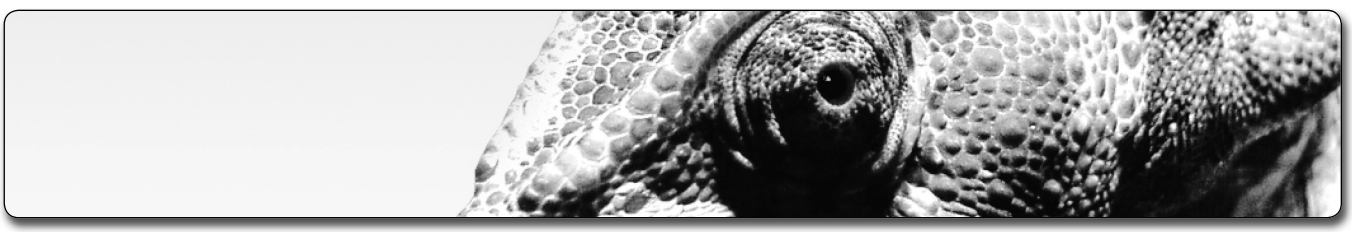
O termo Programação Procedural (ou programação procedimental) é às vezes utilizado como sinônimo de Programação Imperativa (paradigma de programação que especifica os passos que um programa deve seguir para alcançar um estado desejado), mas o termo pode se referir (como neste artigo) a um paradigma de programação baseado no conceito de chamadas a procedimentos. Procedimentos, também conhecidos como rotinas, sub-rotinas, métodos, ou funções simplesmente contêm um conjunto de passos computacionais a serem executados. Um dado procedimento pode ser chamado a qualquer hora durante a execução de um programa, inclusive por outros procedimentos ou por si mesmo.

Listagem 1. Polimorfismo no Delphi Win32

```
unit SeresVivos;

interface

type
  TServivo = class
    procedure ProduzirSom;
  end;

  TCachorro = class(TServivo)
    procedure ProduzirSom;
  end;

  THomem = class(TServivo)
    procedure ProduzirSom;
  end;

implementation

uses
  SysUtils, Dialogs;

{ TCachorro }

procedure TCachorro.ProduzirSom;
begin
  ShowMessage('Cachorro Latindo !!!');
end;

{ TServivo }

procedure TServivo.ProduzirSom;
begin
  raise Exception.Create('Nem todo ser vivo produz
    som !');
end;

{ THomem }

procedure THomem.ProduzirSom;
begin
  ShowMessage('Homem Falando !!!');
end;

end.

procedure TForm1.Button1Click(Sender: TObject);
var
  Ser: TSerVivo;
begin
  Ser := THomem.Create;
  Ser.ProduzirSom;
end;
```

Listagem 2. Métodos Virtuais

```
unit SeresVivos;

interface

type
  TServivo = class
    procedure ProduzirSom;virtual;
  end;

  TCachorro = class(TServivo)
    procedure ProduzirSom;override;
  end;

  THomem = class(TServivo)
    procedure ProduzirSom;override;
  end;

implementation

end.
```



precisamos declarar o tipo de nossas variáveis, não há porque os métodos serem estáticos, e nem podem. Como não há um tipo definido para o objeto que será instanciado, por padrão será chamado o método do objeto que foi instanciado para aquela variável. Por isso podemos dizer que no PHP os métodos são "Virtuais" e isto facilita a implementação do *polimorfismo* reduzindo assim o uso de controles de fluxos como *If's* para tornar nosso código mais robusto e expansível.

Você deve estar se perguntando: *"Se não precisamos declarar o tipo de nossas variáveis e de nossos objetos, como saber se um objeto pertence a uma determinada hierarquia ou possui o método que queremos chamar?"*.

Para isto temos no PHP 5 o operador *instanceof***.** Ele verifica se um objeto qualquer possui um relacionamento *é um* com uma classe específica. Ele na verdade é uma modificação da já existente função *is_a()* que foi descontinuada, pois o primeiro é um operador binário lógico permitindo criar sentenças como a seguir:

```
if($c instanceof Cliente) {
  echo '$c é um cliente';
}
```

Para exemplificar essa facilidade observe o código da **Listagem 3**.

Podemos notar facilmente que o código escrito não está expansível. Imagine que você decida criar em seu modelo mais cinco animais cada um produzindo seu respectivo som, isso se tornaria para você um tormento, pois teria que adicionar outros cinco blocos *elseif* para fazer a verificação. Isso fere o conceito de reaproveitamento de código e pior do que isso imagine o sofrimento que seria para dar manutenção em um código deste porte. Então esse código está totalmente fora de cogitação.

O uso da herança com o *polimorfismo* nos ajuda a resolver este problema, a intenção é fazer a mesma coisa, porém de uma maneira que nos ajude a expandir nosso código, reaproveitá-lo e mais do que isso que facilite a manutenção do mesmo. Isso aplicado na prática se torna o código descrito na **Listagem 4**.

Note que agora criamos uma classe *SerVivo* que servirá de base para todas as outras classes de animais que iremos criar e que irão produzir algum som. É nesta classe que está declarado o método que será sobrescrito nas classes descendentes porém não há aqui a necessidade de nenhum parâmetro ou diretiva para tornar este método virtual e passível de ser sobrescrito. Observe que desta maneira o nosso código não fica "engessado". Se por exemplo decidirmos criar uma classe *Gato* em nada teremos que alterar o método *GetSom*. Basta que a classe *Gato* herde de *SerVivo* para que seja aceita no método *GetSom* e feito isso sobrescrevemos o método *ProduzirSom* na classe herdada.

Perceba que foi utilizado em nosso código o operador *instanceof* para verificar se o objeto passado como parâmetro para o método *GetSom* pertence à classe *SerVivo* ou herda da mesma. Se esta condição for satisfeita podemos ter a certeza que ele terá em sua estrutura o método *ProduzirSom*. Agora qual mensagem será exibida por este método vai depender do tipo do objeto que estiver instanciado ali no momento. Com isso voltamos à definição do início do nosso artigo: *"Polimorfismo: a capacidade que um mesmo método tem de se comportar de maneira diferente dependendo de qual classe ele foi chamado"*.

Claro que muito mais pode ser feito para tornar nosso código mais coeso, por exemplo, se nem todos os seres vivos produzem som não há porque o método ProduzirSom na classe SerVivo tenha implementação. Neste caso aplica-se um método abstrato.

Métodos e Classes Abstratas

No PHP5 temos também o conceito de classes e métodos abstratos. Diz abstrato um método que não possui implementação ou uma classe que não deve ser instanciada, que serve apenas de referência para ser herdada e o método para ser sobrescrito.

Isto é muito comum em OO, pois quando trabalhamos com grandes equipes e em grandes projetos geralmente queremos garantir que determinadas classes tenham certos métodos essenciais ao funcionamento do modelo e como são vários os programadores envolvidos no processo definimos esses métodos como abstratos demonstrando assim a nossa intenção de que ele seja sobrescrito na classe que o herdou. Com isso temos um código mais organizado, mais coeso. A própria classe nos lembra que precisamos implementar esse método em algum momento. No BDS 2006, quando temos um método abstrato em uma classe e herdamos da mesma, ao utilizarmos o *CodeCompletion* este método aparece em vermelho indicando que precisa ser implementado.

Um fato importante a ser lembrado é que, no PHP se uma classe possui ao menos um método abstrato ela deve ser declarada como abstrata diferente de outras linguagens de programação. Essa dupla definição existe para permitir a você declarar uma classe como abstrata mesmo que a mesma não possua métodos abstratos.

No exemplo dos nossos seres vivos fica claro que nem todos os seres produzem som, assim este método nesta classe não deve ter implementação ficando nossa classe como a da **Listagem 5**.

Note na **Listagem 5** que a classe *SerVivo* agora foi declarada como *abstract*, pois dos três métodos que a mesma possui um deles é abstrato e por isso temos que declarar a classe como abstrata também. Assim *SerVivo* não poderá ser instanciada e também não haverá necessidade, pois esta classe é apenas uma base para as demais.

Como todo ser vivo nasce, cresce, se reproduz e morre, podemos observar que temos algumas dessas fases em forma de métodos em *SerVivo*, não como abstratos, mas passíveis de serem sobrescritos.

Métodos e Propriedades Estáticas

É de conhecimento geral que uma classe pode declarar "n" propriedades e métodos. Cada instância desta classe, ou seja, cada objeto possui uma cópia destas propriedades e métodos. Isso significa que podem receber valores distintos e seus métodos serem chamados de forma independente. Porém em OO temos um conceito chamado de propriedades estáticas e métodos estáticos. São elementos que pertencem à classe e não ao objeto, com isto eles podem ser acessados e invocados a partir da classe sem a necessidade de se instanciar o objeto.

Podemos ter vários objetos, mas todos eles oriundos de uma única classe. Como as propriedades e métodos pertencem à classe podemos deduzir que essas propriedades e métodos são compartilhados por todos os objetos.

Há algumas considerações sobre o uso de métodos e propriedades estáticas no PHP. Por exemplo, para se acessar propriedades estáticas de dentro da própria classe temos que utilizar o operador *Self::* que aqui tem uma função oposta a do Delphi onde o *Self* é usado para fazer referência ao objeto do contexto enquanto no PHP usamos para acessar a classe. Outro detalhe é que para acessar uma propriedade ou método deste tipo temos que fazê-lo através da classe de duas maneiras:

```
Classe::Propriedade
Classe::Método()
```

Mas você deve estar se perguntando se realmente fará uso deste tipo de recurso e se ele é realmente necessário. Pois então observe o código da **Listagem 6**.

O que temos aqui é uma classe cliente comum e simples, porém com um contador que registra a quantidade de objetos desta classe que foram instanciados. Por isso a propriedade estática *$Contador*, propriedade esta que pertence à classe e não ao objeto, é compartilhada por todas as instâncias. Observe que no construtor acessamos esta propriedade através do operador *Self* e o incrementamos em um. Feito isso atribuímos o valor do contador à propriedade *$id*, essa sim pertencente ao objeto.

O método *CriarClientes* executa um laço *for* e instancia dez objetos *Clientes* e logo após mostra no navegador o valor da propriedade *$id*.

Poderíamos incluir nesta classe *Cliente* um método para obter o valor atual do contador da classe. Este método deverá ser estático para evitar que se tenha que instanciar a classe para invocar o método. Veja como ficaria com essa modificação na **Listagem 7**.

Observe que adicionamos a nossa classe *Cliente* um método estático que retorna a posição atual do contador. Para invocá-lo utilizamos a própria classe sem a necessidade de se instanciar um objeto: *Cliente::GetCount()*.

Criando um exemplo

É notório que os exemplos citados acima servem apenas para exemplificar os conceitos abordados aqui. Para que você possa ter uma noção de como o polimorfismo auxilia em tarefas diárias vamos construir um pequeno exemplo abordando o conceito acima. Abra o Delphi for

Listagem 3. Codigo PHP sem polimorfismo

```
<?php
  class Cachorro{
    function Latir(){
      echo “Cachorro Latindo !”;
    }
  }
  class Homem{
    function Falar{
      echo “Homem Falando !”;
    }
  }
  function ProduzirSom($Obj){
    if ($Obj instaceof Cachorro){
      $Obj->Latir();
    }elseif($Obj instanceof Homem){
      $Obj->Falar();
    }else{
      echo “O objeto passado não produz som”;
    }
  }

?>
```

Listagem 4. Polimorfismo aplicado no PHP

```
<?php
  class SerVivo{
    function ProduzirSom(){
      echo “Nem todo ser vivo produz som !”;
    }
  }
  class Cachorro extends SerVivo{
    function ProduzirSom(){
      echo “Cachorro Latindo !”;
    }
  }
  class Homem extends SerVivo{
    function ProduzirSom{
      echo “Homem Falando !”;
    }
  }
  function GetSom($Obj){
    if ($Obj instaceof SerVivo){
      $Obj->ProduzirSom();
    }else{
      echo “O objeto passado não é um ser vivo !”;
    }
  }
?>
```

Listagem 5. Classes abstratas no PHP

```
<?php
  abstract class SerVivo{
    function Nascer(){
      echo “Nascendo !!!”;
    }
    function Crescer(){
      echo “Crescendo !!!”;
    }
    abstract function ProduzirSom();
  }

  class Cachorro extends SerVivo{
    function ProduzirSom(){
      echo “Cachorro Latindo !”;
    }
  }

  class Homem extends SerVivo{
    function ProduzirSom{
      echo “Homem Falando !”;
    }
  }
  function GetSom($Obj){
    if ($Obj instaceof SerVivo){
      $Obj->ProduzirSom();
    }else{
      echo “O objeto passado não é um ser vivo !”;
    }
  }
?>
```

Listagem 6. Propriedade Estática no PHP

```
<?php
  class Cliente{
    static $Contador = 0;
    public $id;
    function __construct(){
      self::$Contador++;
      $this->id = self::$Contador;
    }
  }
  function CriarClientes(){
    for($i=0;$i<=9;$i++){
      $cliente = new Cliente();
      echo $cliente->id.”<br>”;
    }
  }
?>
```

PHP e crie uma nova aplicação. Para isso acesse o menu *File|New>Application*.

Salve a aplicação pressionando *Ctrl+Shift+S* ou escolha *File>Save Project As* com o nome "PrjPolimorfismo.phprj" e em seguida salve a *Unit* como "index.php". Altere a propriedade *Name* do formulário para "FrmIndex". Este formulário será utilizado para criar a *interface* visual do nosso exemplo.

Adicione neste formulário três *Edits* da paleta *Standard*. Altere seus nomes para "EdtNome", "EdtEmail", "EdtRegistro", respectivamente. Adicione três *Labels*, também da paleta *Standard*, um para cada *Edit* e troque seus *Captions* de acordo com cada campo. Adicone também dois Buttons("BtnSalvar" e "BtnListar") e um *RadioGroup*. Adicione ao *RadioGroup* dois itens a sua propriedade *Items*. Digite "Pessoa Física" e "Pessoa Jurídica". Altere a propriedade *Orientation* para *orHorizontal*. Na **Figura 1** temos uma sugestão de *layout*.

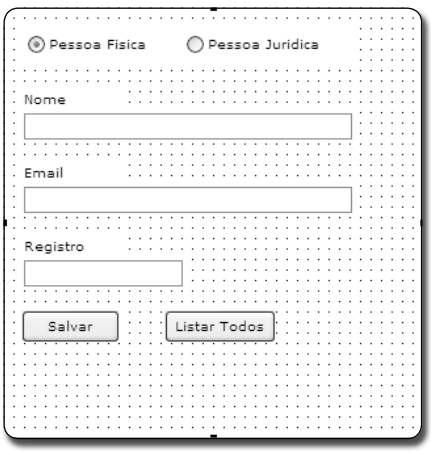


Figura 1. Exemplo de layout

Concluído o layout passaremos agora para a criação de nossas classes. Não irei me apronfundar na sintaxe OO no PHP pois este foi escopo do primeiro artigo.

Acesse o menu *File|New>Unit* e crie uma nova *Unit* salvando-a com o nome "Model.php". Nesta *Unit* iremos criar um modelo onde teremos uma classe *Pessoa* com as propriedades *Nome* e *Telefone* comum a todo o tipo de pessoa. Faremos um especialização de *Pessoa* em *PessoaFisica* e *PessoaJurica* cada uma com uma propriedade exclusiva *CPF* e *CNPJ* respectivamente. Na *Unit* criada digite o código da **Listagem 8**.

No código da **Listagem 8** apenas criamos a estrutura das classes que serão usadas em nosso exemplo. Posicione o cursor na área da classe *Pessoa* e pressione *Ctrl + Shift + Alt + U* para publicar as propriedades. No diálogo que se abre preencha os campos conforme a **Figura 2.**

Repare que o Delphi for PHP já adiciona à classe um campo protegido *$_nome* com os métodos *Get* e *Set* para acesso a este *Field*. Isso fará que no *Code Completion* você possa acessar a propriedade *Nome* através de *Pessoa->Nome* embora o *GetNome* e o *SetNome* não sejam invocados.

Retornando ao exemplo, repita o passo anterior e faça o mesmo para publicar em *Pessoa* a propriedade *Telefone*. Em *PessoaFisica* publique a propriedade *CPF* e em *PessoaJuridica* a propriedade *CNPJ*. Observe na **Listagem 10** as classes com as propriedades publicadas.

A princípio você deverá notar que não foge muito a regra de criação de classe em comparação com o Delphi Win32. Temos em *Pessoa* as propriedades pertinentes a todo tipo de pessoa. *PessoaFisica* herda de *Pessoa*, e por esse motivo traz consigo, devido a herança, as propriedades *Nome* e *Telefone*. Precisamos apenas adicionar a propriedade *CPF* assim como acontece em *PessoaJuridica*, sendo que nesta temos o *CNPJ* como diferencial.

Se notou bem, publicamos o método *Save()* em *Pessoa* como *abstract* indicando assim a intenção que temos em sobrescrevê-lo nas classes descendentes e é isso que faremos agora. Na classe *PessoaFisica* sobrescreva a função *Save()* conforme a **Listagem 11**. Faça o mesmo para *PessoaFisica* conforme **Listagem 12**.

Listagem 7. Método Estático no PHP

```
<?php
  class Cliente{
    static $Contador = 0;
    public $id;
    function __construct(){
      self::$Contador++;
      $this->id = self::$Contador;
    }
    static function GetCount(){
      return self::$Contador;
    }
  }
  function CriarClientes(){
    for($i=0;$i<=9;$i++){
      $cliente = new Cliente();
      echo $cliente->id."<br>";
    }
       echo Cliente::GetCount();
  }

?>
```

Listagem 8. Estrutura das classes

```
  abstract class Pessoa{
    protected abstract function Save();
  }
  class PessoaFisica extends Pessoa{
  }
  class PessoaJuridica extends Pessoa{

    }
```

Nota do DevMan

Para que os métodos Get e Set possam ser invocados automaticamente como no Delphi Win32 suas classes precisam possuir os métodos __set() e __get ou simplesmente herdar suas classes da classe Object do Delphi for PHP o que não acontece automaticamente como no caso so Delphi Win32. A classe object já intercepta o acesso a propriedade e delega a chamada aos respectivos get's e set's. Oberve na Listagem 9 a estrutura em Object.

Declaramos os métodos __get() e __set() e preparamos ambos para que localizem a herança e retornem o resultado. Caso não seja encontrada, uma exceção é levantada informando que o método não existe.

Figura 2. Publicando a Propriedade Nome

Esses são os métodos que serão fruto do *polimorfismo*. Teremos uma outra classe que servirá como lista de pessoas, onde guardaremos todos os objetos criados na aplicação. O botão listar servirá para percorrer toda a lista e invocar o método *Save()* dos objetos contidos nela. Não sabemos ao certo se são pessoas físicas ou jurídicas. Apenas sabemos que são pessoas e se são pessoas possuem o método *Save()*.Na mesma *Unit* do modelo, *model.php*, crie a classe *ListaPessoa* conforme **Listagem 13**.

Nesta classe temos um *Array Private* onde guardaremos todos os objetos criados na aplicação. O método público *AddPessoa* serve para que possamos ter uma maneira de adicionarmos itens ao *Array* que é privado. Repare no parâmetro da função que colocamos o tipo do parâmetro antes do mesmo. O PHP não é tipado, porém temos este recurso que é chamado de *Hint de Classe*.

Com isso um erro será gerado se tentarmos adicionar no *Array* um objeto que não seja do tipo pessoa. A função publicar em *ListaPessoa* apenas faz um *loop* no *Array* onde cada interação, ou seja, cada vez que passa por um objeto o método *Save()* é invocado, publicando no *browser* as informações do objeto.

Para testar nosso modelo vamos ao evento *OnClick* do botão *Salvar*. Neste evento digite o código da **Listagem 14**.

O procedimento é simples. Primeiro verificamos a opção selecionada no *RadioGroup* através da propriedade *ItemIndex*. Feito isso criamos o objeto de acordo com a opção do usuário e já carregamos o valor do *edtRegistro* para a propriedade *CPF* ou *CNPJ* dependendo do caso.

Em seguida carregamos para as propriedades *Nome* e *Email* os valores de seus respectivos *Edit's*. Por fim verificamos se há na seção uma variável *Obj*. Se ela não existir criamos um objeto e o colocamos na seção para poder passá-lo para outra página. Se a variável de seção já existir então apenas invocamos de dentro da seção a função *AddPessoa* passando o objeto criado. Ao final teremos na seção um objeto *ListaPessoa* com todas as pessoas criadas a cada clique de botão.

Para concluir nosso exemplo no *OnClick* do botão *Listar Todos* chamaremos uma página ("lista.php") e nela exibiremos o conteúdo de nossa lista de pessoas. Para isso codifique o botão em questão fazendo uma chamada ao método *redirect* do PHP, conforme a seguir:

```
redirect('lista.php');
```

Esta página ainda não foi criada então proceda com a criação em *File|New>Form*. Salve-a como "lista.php" e modifique

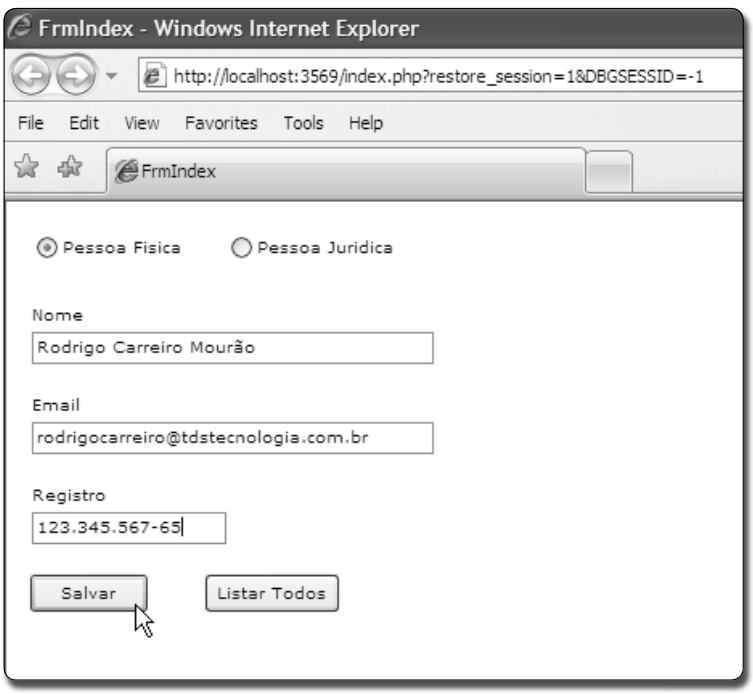


Figura 3. Cadastrando Pessoas na Lista

Listagem 9. Métodos __get e __set em Object.

```
<?php
function __get($nm){
  $method='get'.$nm;
  if (method_exists($this,$method)){
    return ($this->$method());
  }else{
    $method='read'.$nm;
    if (method_exists($this,$method)){
      return ($this->$method());
    }else{
      if ($this->inheritsFrom('Component')){
        if( isset($this->_childnames[$nm]) )
          return $this->_childnames[$nm];
        }
        throw new EPropertyNotFound(
          $this->ClassName().".".$nm);
      }
    }
  }
function __set($nm, $val){
  $method='set'.$nm;
  if (method_exists($this,$method)){
    $this->$method($val);
  }else{
    $method='write'.$nm;
    if (method_exists($this,$method)){
      $this->$method($val);
    }else{
      throw new
EPropertyNotFound($this->
        ClassName().".".$nm);
    }
  }
}
?>
```

Listagem 10. Classes com propriedades publicadas

```
<?php
  /* Classe principal Pessoa */
  abstract class Pessoa{
    protected $_nome="";
    function getNome(){
      return $this->_nome;
    }
    function setNome($value){
      $this->_nome=$value;
    }
    function defaultNome(){
      return;
    }
    protected $_email="";
    function getEmail(){
      return $this->_email;
    }
    function setEmail($value){
      $this->_email=$value;
    }
    function defaultEmail(){
      return;
    }
    protected abstract function Save();
  }
  /* Classe PessoaFisica */
  class PessoaFisica extends Pessoa{
    protected $_cpf="";
    function getCPF(){
      return $this->_cpf;
    }
    function setCPF($value){
      $this->_cpf=$value;
    }
    function defaultCPF(){
      return;
    }
  }
  /* Classe PessoaJuridica */
  class PessoaJuridica extends Pessoa{
    protected $_cnpj="";
    function getCNPJ(){
      return $this->_cnpj;
    }
    function setCNPJ($value){
      $this->_cnpj=$value;
    }
    function defaultCNPJ(){
      return;
    }
  }
?>
```

Listagem 11. Metodo Save() na classe PessoaFisica

```
public function Save(){
  echo '<font size=6 color=darkblue'.
    'face=Tahoma>'.$this->Nome.'</font><br>'.
    'Email...: '.$this->Email.'<br>'.
    'CPF....: '.$this->CPF.'<br>'.
    'Tipo...: Pessoa Fisica <hr>';
}
```

Listagem 12. Metodo Save() na classe PessoaJuridica

```
public function Save(){
  echo '<font size=6 color=darkblue'.
    'face=Tahoma>'.$this->Nome.'</font><br>'.
    'Email...: '.$this->Email.'<br>'.
    'CNPJ...: '.$this->CNPJ.'<br>'.
    'Tipo...: Pessoa Juridica <hr>';
}
```

Listagem 13. Classe Pessoa Lista

```
  class ListaPessoa{
    static private $instance;
    private $_list = array();

    function AddPessoa(Pessoa $Obj){
      $this->_list[] = $Obj;
    }

    function Publicar(){
      for ($i = 0; $i < count($this->_list); $i++){
      $this->_list[$i]->Save();
    }
  }
```

Listagem 14. Criando os objetos do modelo

```
function BtnSalvarClick($sender, $params){

  switch ($this->RadioGroup1->ItemIndex) {
    case 0:
      $Pessoa = new PessoaFisica;
      $Pessoa->CPF = $this->Edit3->Text;
      break;
    case 1:
      $Pessoa = new PessoaJuridica;
      $Pessoa->CNPJ = $this->Edit3->Text;
  }

  $Pessoa->Nome  =  $this->Edit1->Text;
  $Pessoa->Email =  $this->Edit2->Text;

  if (!isset($_SESSION['Obj'])){
    $_SESSION['Obj'] = new ListaPessoa;
  }

  $_SESSION['Obj']->AddPessoa($Pessoa);
}
```

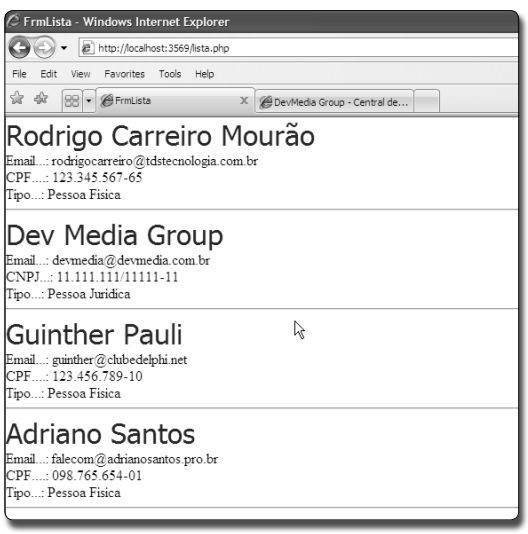


Figura 4. Publicando conteúdo da lista

seu *Name* para "FrmLista". No *OnShow* dessa página digite o código a seguir:

```
$L = $_SESSION['Obj'];
$L->Publicar();
```

Este código apenas invoca o método *Publicar* do objeto *ListaPessoa* que está na seção. Este método por sua vez faz um *loop* no *Array* invocando o método *Save()* de cada item da lista. Com isso quando esta página for carregada exibirá no *browser* todas as pessoas cadastradas na lista devidamente formatadas.

Execute a aplicação. *Selecione* o tipo de pessoa, informe os dados e clique no botão *Salvar*. Veja a aplicação em execução nas **Figuras 3** e **4**.

Conclusão

Com os conceitos aprendidos até o momento já é possível adicionar aos nossos projetos um aspecto mais profissional, e claro, facilitar bastante o nosso trabalho. Confesso que para quem não tem um certo conhecimento em OO pode parecer a primeira vista um tanto complicado aplicar todos esses conceitos, mas com o tempo você irá perceber que as coisas não são tão complicadas como parecem. O mundo de programação OO é fascinante e requer muita dedicação por parte daqueles que se lançam nele. Por isso dedique-se. Lembre-se que OO é OO independente da linguagem. Habitue-se a pensar OO, pois todas as grandes linguagens e por conseqüência ferramentas estão apoiadas nesta filosofia.

Um grande abraço a todos e até o próximo artigo, onde trataremos de um assunto um tanto interessante: Design Patterns em PHP.

Eu sou Rodrigo Carreiro e pela sua atenção muito obrigado. ●

Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/javamagazine/feedback

Já pensou em hospedar todos os seus sites em um único plano de hospedagem?

PLANO PROFISSIONAL 1

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe!

RVSTMEDIA

- 3 domínios independentes
- 50 GB de transferência
- 30 GB de e-mails com SSL
- 1 GB de espaço
- Painel de controle
- ASP, ASP.NET 3.5 E PHP 5
- Access ilimitado
- 3 bancos MYSQL 5
- 1 banco SQL Server 2005 Express

Tudo isso por apenas R$ **25,90** por mês

Na Hospedix você ainda ganha registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

Acesse agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

Quantas cópias de seu software existem no mercado?

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700